TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Gleb Engalychev 206721IADB

Sõltuvuste ühilduvuse analüüsitööriista arendus Spring raamistiku versiooniuuenduste jaoks

Bakalaureusetöö

Juhendaja: Jaanus Pöial
PhD

Tallinn 2024

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: Gleb Engalychev

03.01.2024

Annotatsioon

Iga tõsine tarkvaraarenduse projekt kasutab kolmandate osapoolte tehnoloogiaid. Need tehnoloogiat on pidevalt uuendatud. Tehnoloogiate uuendused sisaldavad turvalisuse, vigade ja jõudlusi parandusi ning uusi võimekusi. Selleks et saada väärtust nendest uuendustest on vaja tarkvaraarenduse projektis kasutatud tehnoloogiate versioone uuendata.

On võimalikud olukorrad, kui tarvaraprojektis kasutatud tehnoloogiaid ei ole uuendatud mitme aasta jooksul ja ühel hetkel otsustatakse teha tehnoloogiate uuendust. Antud olukorras tehnoloogiate uuendamine võib olla mahukas, kuna aastate jooksul tehnoloogiad muutusid ja nende uuemaid versioone on vaja teistmoodi kasutada võrreldes vanade versioonidega.

Antud töö raames on käsitletud probleeme, mis on seotud vana tarkvaraarenduse projekti tehnoloogiate uuendamise tööde planeerimisega. Probleemide lahendamiseks otsustatakse arendada uut rakendust. Rakenduse jaoks on kehtestatud funktsionaalsed ja mittefunktsionaalsed nõuded. Kehtestatud nõuete alusel valitakse vajalikke tehnoloogiaid rakenduse arenduseks ja disainitakse rakenduse disaini prototüübi. Valitud tehnoloogiate ja disaini prototüübi alusel on arendatud uus rakendus.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 31 leheküljel, 5 peatükki, 10 joonist, 5 tabelit.

Abstract

Development of Dependency Compatibility Analysis Tool for Spring Framework Version Upgrades

Every serious software development project uses third-party technologies. These technologies are constantly updated, with updates including improvements in security, bug fixes, performance improvments and new features. In order to derive value from these updates, it is necessary to upgrade the versions of the technologies used in the software development project.

There are possible situations where the technologies used in a software project have not been updated for several years and at some point a decision is made to update these technologies. In this situation updating the technologies can be cumbersome, as over the course of several years the technologies have evolved and their newer versions need to be used differently compared to the old versions.

This work addresses issues related to planning the update of technologies in an old software development project. To solve these problems, a decision is made to develop a new application. Functional and non-functional requirements are established for the application. Based on these requirements, the necessary technologies for application development are chosen and a prototype of the application's design is created. Using the selected technologies and the design prototype as a basis, a new application is developed.

The thesis is in estonian and contains 31 pages of text, 5 chapters, 10 figures, 5 tables.

Lühendite ja mõistete sõnastik

API	API on tarkvarakomponentide kogum, mis võimaldab erinevatel rakendustel omavahel suhelda. See määratleb, kuidas tarkvara komponendid peaksid üksteisega suhtlema, võimaldades seeläbi andmete ja funktsionaalsuste vahetamist.
CLI	CLI on kasutajaliides, mis võimaldab kasutajal suhelda arvutiprogrammiga tekstipõhiste käskude abil. See toimib käsurea kaudu, kus kasutaja sisestab käske tekstina ja saab vastuseid samuti teksti kujul.
Github	GitHub on veebipõhine platvorm, mis pakub arendajatele võimalust hoida, hallata ja jagada oma tarkvaraprojektide koodi, võimaldades koostööd ning versioonihaldust. See integreerib endas kodeerimist, probleemide jälgimist, pull-päringuid ja automatiseeritud ehitamist, muutes tarkvaraarenduse efektiivsemaks ja läbipaistvamaks.
GUI	GUI on kasutajaliides, mis võimaldab kasutajal suhelda arvutiprogrammiga graafiliste elementide abil, näiteks ikoonide, nuppude ja akendega. GUI teeb tarkvara kasutamise intuitiivsemaks ja vähem sõltuvaks tekstipõhistest käskudest.
IDE	IDE on tarkvarapakett, mis pakub arendajatele integreeritud keskkonda, kus nad saavad kirjutada, testida, siluda ja hallata oma koodiprojekte. See sisaldab sageli tekstiredaktorit, kompilaatorit, silumisvahendit ja muid arendusseadmeid ühes kohas.
Parsimine	Parsimine on protsess, kus tarkvara analüüsib ja tõlgendab struktureeritud või poolstruktureeritud andmeid. Näiteks võib see hõlmata teksti, koodi või muud sisendit, mida tarkvara töötleb ja jagab osadeks, et mõista selle struktuuri ja tähendust.
Raamistik	Raamistik on struktuur või platvorm, mis pakub üldkasutatavaid tööriistaid ja komponente tarkvara arendamiseks, lihtsustades programmeerijate tööd.
Spring Boot	Raamistik, mis lihtsustab Spring raamistikuga ehitatud rakendusi seadistada ja paigaldada.

Sisukord

Jooniste loetelu

Tabelite loetelu

1 Sissejuhatus

Iga tõsine tarkvaraarenduse projekt kasutab tehnoloogiaid, mis on arendatud koos teiste osapooltega. Kogu kasutatud tarkvara, mis tuleb teistelt osapooltelt, saab oma elutsükli jooksul hulga uuendusi. Need uuendused pakuvad turvalisuse, vigade ja jõudluse parandusi ning uusi võimekusi. Selleks, et saada kasu tehnoloogiate uuendustest, projekt mis kasutab teiste osapoolte tehnoloogiaid peab uuendama kasutatud tehnoloogiate versioone. Juhul kui tehnoloogia uuendusega ei ole muudetud viisi, kuidas seda tehnoloogiat on vaja kasutada, toimub tehnoloogia uuendus kergelt. Juhul, kui tehnoloogia jaoks on tehtud mahukas uuendus, mis muudab viisi, kuidas tehnoloogiat on vaja kasutada, võib tehnoloogia uuendus olla keeruline. Antud stsenaariumil tuleb muuta projekti, et see kasutaks tehnoloogiat uuel viisil.

Mõnikord on võimalikud olukorrad, kui tarkvaraarenduse projektis kasutatud teiste osapoolte tehnoloogiaid ei ole uuendatud mitme aasta jooksul. Kuna tehnoloogiate uuendus võtab lisa arendusaega, on võimalikud olukorrad, et juhtkond otsustab mitte kulutada ressursse tehnoloogiate uuendustele, kuna see ei anna otseset kasu tarkvara kasutajatele. Esialgu selline olukord ei ole kahjulik tarkvara projekti jaoks. Mingi hetk juhtkond ikkagi otsustab teha tehnoloogiate uuenduse mingil põhjusel, nt. kui tarkvaraarenduse projektis kogunes liiga palju probleeme turvalisusega või ei ole võimalik implementeerida uut funktsionaalsust mõistliku aja jooksul ilma tehnoloogia uue funktsionaalsuseta.

Juhul, kui tarkvaraarenduse projekti tehnoloogiad ei ole uuendatud mitme aasta jooksul, on nende uuendamine värskete versioonideni mahukas töö.

Antud töö raames on arendatud tarkvaralahendus, mille eesmärk on lihtsustada vanade rakenduste tehnoloogiate uuendamise protsessi planeerimist.

2 Probleemi kirjeldus ja taust

Selles peatükis on detailselt kirjeldatud töös lahendatud probleemi ja selle tausta. On kirjeldatud infosüsteemi ja tööprotsesse, mille kontekstis probleem esineb. On valitud probleemi lahendus ning on antud põhjused, miks lahendus oli valitud.

2.1 Olemasoleva infosüsteemi ülevaade

Infosüsteem koosneb paljudest rakendustest, mis töötavad ja suhtlevad omavahel. Rakendused on ehitatud kasutades erinevaid tehnoloogiaid ja rakendusi on palju erinevaid tüüpe. Antud töö raames on oluline ainult teada, et koodibaasis eksisteerib vähemalt 50 rakendust, mis on ehitatud kasutades Spring raamistikku ja Java programmeerimiskeelt. Koodibaas eksisteerib umbes 30 aastat ja pideva arengu jooksul koodibaasis olid arendatud ühised teegid. Teekide eesmärgiks on ühiste rakenduste komponentide standardiseerimine ja korduvkasutamine. Ühiste teekide näited: autentimise teek, vigade halduse teek, isikute halduse teek.

2.2 Probleemi kirjeldus

Töö autor töötab firmas, kust enne mainitud infosüsteemi haldaja tellib arendustööd. Üks arendustöö mida tihti tellitakse on Spring raamistikuga ehitatud rakenduses kasutatud tehnoloogiate uuendamine. Antud kontekstis tehnoloogiate uuendamine tähendab projektis kasutatud programmeerimiskeele uuema versiooni kastuselevõtmine, Spring ja Spring Boot'i raamistike uuema versiooni kasutuselevõtmine, projektis kasutatud ehitustööriista uuema versiooni kasutuselevõtmine ning projektis kasutatud teekide uuemate versioonide kasutuselevõtmine. Tehnoloogia uuema versiooni kasutuselevõtmine tähendab seda, et Spring rakenduses on tõstetud suvalise tehnoloogia versioon ja pärast seda rakendus on täiendatud ja muudetud niimoodi, et see töötab korrektselt uuema tehnoloogia versiooniga.

Enne tehnoloogiate uuendamise tellimuse tegemist klient palub anna firmat kus töö autor töötab tehnoloogiate uuendamiseks vajalikele töödele mahuhinnangut. Mahuhinnangu

andmisega tegeleb tarkvaraarendaja. Mahuhinnagu andmiseks tarkvaraarendaja peab hinnatava rakenduse koodi analüüsima. Analüüsi jooksul on vaja leida need rakenduse osad, mis ei ole ühilduvad uuema tehnoloogia versiooniga. Rakenduse osa ei ole ühilduv uuema tehnoloogia versiooniga juhul, kui tehnoloogia funktsionaalsus mis on kasutatud rakenduse osas on eemaldatud või muudetud uuemas tehnoloogia versioonis. Ebaühilduvuse uuema tehnoloogia versiooniga näidiseks on rakenduses teegi meetodi kasutamine, mis uuemas teegi versioonis sai eemaldatud.

Rakenduse tehnoloogiate uuenduse mahuhinnangu andmiseks analüüsi tegemise keeruline osa on selles peatükkis esimeses lõigus mainitud ühiste teekide analüüs. Paljud ühised teegid kasutavad Spring raamistiku funktsionaalsust. Ühiste teekide integreerimisel rakendusse teegid hakkavad kasutama rakendusega koos tuleva Spring raamistiku funktsionaalsust. Sellest tuleb järeldus, et kui enne mainitud ühine teek on kasutatud rakenduses, see peab olema ühilduv rakenduses kasutatavaga Spring raamistikuga.

Töö kirjutamise hetkel ühiste teekide hindamine võtab palju aega ja sõltub palju inimfaktorist. Tavaliselt arendaja, kes tegeleb rakenduse analüüsiga, ei ole tuttav ühise teegi koodiga nii hästi nagu rakenduse koodiga kus uuendus toimub. Sellepärast arendajale on vaja kulutada lisaega, et teha endale ühise teegi koodi selgemaks. Uuendatava rakenduse koodiga tarkvaraarendaja on tavaliselt tuttav sellepärast, et infosüsteemi haldajalt rakenduse tehnoloogiate uuendamise tellimusega koos tavaliselt tuleb rakenduse täiendava arenduse tellimus, mille täitmise jooksul tarkvaraarendaja teeb endale rakendust selgemaks. Ühistel teekidel puudub ka dokumentatsioon, mis teeb koodi analüüsi keerulisemaks. Veel ei ole alati täpselt teada, mis Spring raamistiku osad said eemaldatud uues Spring versioonis. Spring raamistiku arendajate poolt on koostatud migreerimise juhendid Spring versioonidele 5.0.0.RELEASE kuni 5.3.x [1] ja versioonidele 6.0.0 kuni 6.1.x [2], aga nad ei anna tervet listi sellest, mis osad raamistiku APIst said täpselt muudetud või eemaldatud. Mainitud põhjuste tõttu arendaja kulutab palju aega analüüsile ning on tõenäoline ka see, et analüüsitava koodi ebateadmise tõttu arendaja teeb viga oma mahuhinnagu andmisel.

Selle konteksti alusel on vormuleeritud järgmine probleem: Spring sõltuvuste versioonide uuendustega seotud tööle mahuhinnangu andmine võtab palju aega ja sõltub suures määras inimfaktorist.

2.3 Probleemi lahendusmetoodika valimine

Kõige kindlamaks probleemi lahenduseks oleks infosüsteemi haldaja poolt kirjeldatud infosüsteemi ühiste teekide pidev toetus, et ühised teegid on alati ühilduvad kõige viimase Spring raamistiku versiooniga. Sellel juhul rakenduse tehnoloogiate uuendamisele mahuhinnagu andmisel ei oleks vaja rakenduses kasutatud ühiseid teeke analüüsida, sest nad on garanteeritult ühilduvad uuema Spring raamistiku versiooniga. Selline lahendus pareneks ka ühiste teekide turvalisust ja kvaliteetsust, sest uuemad ühiste teekide sõltuvuste versioonid sisaldavad vigade ja turvalisuse parandusi [3]. Praegu infosüsteemi haldajal ei ole ressursse sellele muudatusele ning sellepärast oli otsustatud probleemi lahendamiseks valida erinevat metoodikat.

Teiseks probleemi lahenduseks on rakenduse arendus, mis võimaldab teha automaatset ühiste teekide skaneerimist, mille tulemuseks on visuaalne raport, mis annab ülevaaded kõikdest kohtadest, kus teek ei ole ühilduv uuema Spring rakenduse versiooniga. Selle lahendusega on vähendatud arendaja käsitöö, kui ta analüüsib teekide lähtekoodi ning on vähem sõltuvust inimfaktorist, sest teegi analüüsiga tegeleb programm.

Lahenduse puuduseks on see, et uue rakenduse kirjutamisele tuleb kulutada arendusressursi. Esialgu rakendus oma arendusele kulutatud ressursi ei tasu. Pikemas perspektiivis selline arendus tasub, sest tehnoloogiad pidevalt muutuvad ning nende uuendamine toimub alati. Sellel stsenaariumil arendatud tööriist on aktuaalne ka tulevikus, kui tuleb anda mahuhinnangut tehnoloogiate uuendamise tööle.

2.4 Eksisteerivad lahendused

Töö kirjutamise hetkel alternatiivseid tööriistu püstitatud probleemi lahendamiseks ei eksisteeri. Seotud teemaga oli leitud üks tööriist, mis aitab migreerida vana Spring rakendust uuemale Spring versioonile. Tööriista nimi on Spring Boot Migrator ja see on arendatud Spring raamistiku arendustiimina. Tööriist aitab automatiseerida vanade Spring rakenduste migratsiooni Spring Boot raamsitikule. [4]

3 Arendatava rakenduse analüüs ja projekteerimine

Rakenduse arendus on jagatud kolmeks põhiliseks etappiks: analüüs, arendus ja testimine. Selleks et arusaadavalt ja täpselt defeneerida mida ja mis raames rakendus peab tegema defeneeritakse funktsionaalseid ja mittefunktsionaalseid nõudeid. Funktsionaalsed nõuded defineerivad seda, mida süsteem peab tegema ning mittefunktsionaalsed nõuded seda, kuidas süsteem peab seda tegema [5]. Funktsionaalsete nõuete defineerimiseks on kasutatud *user stories* ehk kasutajalood. Pärast seda kui nõuded on defineeritud, on tehtud rakenduse projekteerimine ning rakenduses kasutatavate tehnoloogiate valik.

3.1 Funktsionaalsed nõuded.

Töö kirjutamise hetkel infosüsteemi ühised teegid ei ole ühilduvad kõige tihedamini Spring versiooniga 6 ning on ühilduvad Spring versiooniga 5. Samuti hinnang kas teek on ühilduv Spring versiooniga 6 või mitte pakkub kõige rohkem keerukust ja võtab kõige rohkem aega, võrreldes hinnangutega teistele Spring ramistiku versioonide uuendamistele. Sellepärast rakenduse skoobiks on teha tööriistat, mis annab ülevaadet kas teek on ühilduv Spring versiooniga 6 või mitte. Teiste Spring versioonidega ühilduvuse kontroll on suurem arendus, mis teeb töö skoobi liiga laiaks.

Rakendus peab oskama genereerima visuaalset raporti, mida on võimalik kergesti üles laadida, avada ja teistele inimestele saata. Nõue on põhjendatud sellega, et tarkvaraarendaja peab üle andma ühilduvuse raporti projektijuhile, kes tehnoloogia uuendamise lepingu sõlmimisel kasutab koostatud aruannet uuendamise tööde mahu ja tähtajade põhjendamisel.

Ühilduvuse raporti sisuks on iga koodirida teegis, mis ei ole ühilduv Spring versiooniga 6. Iga ebaühilduva koodirea kohta peab olema näha:

- Mis failis ebaühilduv koodirida asub.
- Mis real ebaühilduv koodirida asub.
- Mis veerul ebaühilduv koodirida asub.

- Mis API osa Spring raamistikust on kasutatud sellel koodireal, mis ei ole ühilduv Spring versiooniga 6.
- Ebaühilduva koodirida staatus Spring raamistikus versiooniga 6. Kas ebaühilduv koodirida said eemaldatud või muudetud uuemas raamistiku versioonis.

Sellest infost peab olema piisav, et iga ebaühilduva koodirea kohta anda hinnangut. Sellega tarkvaraarendaja võib kiiresti teada, kus skannitavas teegis täpselt ebaühilduv kood asub ning mis osa Spring raamistiku APIst on kasutatud.

Rakenduse kasutajaliidese tüüp on CLI liides. Selline valik on tehtud sellepärast, et CLI liidest on lihtsam teha kui GUI liidest ning CLI liidese puudused ei esine arendatava rakenduse kontekstis. CLI kasutajaliidese puuduseks on see et tavalisele kasutajale CLI liides ei ole intuitiivselt arusaadav, nagu GUI tüübi kasutajaliides. Rakenduse kasutamise jaoks on plaanitud koostada kasutamisjuhendit ning rakenduse peamiseks kasutajarühmaks on tarkvarajendajad, kes töötavad igapäevaselt CLI liidestega. Sellepärast ei ole suureks probleemiks, et tavalisele kasutajale, kes ei ole tarkvaraarendaja, CLI liides ei ole intuitiivselt arusaadav ja mugav.

Tehtud analüüsi kokkuvõtteks on võimalik defineerida järgmiseid funktsionaalseid nõudeid. Nõuded on defineeritud kasutajalugude abil:

- Tarkvaraarendajana mina tahan genereerida valitud teegi kohta ühilduvuse raporti Spring raamistiku versiooniga 6.
- Tarkvaraarendajana ja projektijuhina mina tahan, et genereeritud ühilduvuse raporti oleks võimalik mugavalt teistele inimestele saata.
- Tarkvaraarendajana mina tahan saada kõiki vajalikke andmeid teegi ebaühilduvate koodiridade kohta. Selleks iga koodirea kohta raportis peab olema näha: mis teegi failis, faili real ja veerul koodirida asub, mis API osa Spring raamistikust on kasutatud koodireal.
- Tarkvaraarendajana mina tahan, et rakendusel oleks CLI kasutajaliides.

3.2 Mittefunktsionaalsed nõuded

Rakenduse jaoks olid defineeritud järgmised mittefunktsionaalsed nõuded:

- Rakendus peab oleam võimeline genereerima ühilduvuse raporti mitte rohkem kui 1 minuti jooksul.
- Rakenduse kood peab olema kergesti laiendav, testitav ja aru saadav.
- Rakenduse kasutajaliides peab olema mugav ja lihtne.
- Rakenduse jaoks peab olema koostatud kasutamisjuhend.

3.3 Tehnoloogiate valik

Selles peatükkis töö autori poolt on tehtud mitu teeke ja tehnoloogiate valikuid püstitatud probleemi lahendamiseks. Iga teegi valimiseks töö autor teostab võrdlusuuringu, mille jooksul on leitud erinevad sarnased teegid allpeatükis kijreldatud ülesanne lahendamise jaoks. Uuringu jooksul töö autor proovib leida sobivaid tööriistu kasutades otsingu internetist. Teegi valimisel autori poolt on tehtud võrdlusanalüüs, mille käigus ta võrreldab kättesaadavaid lahendusi ja valib üht teegi mis soobib kõige paremini püstitaud ülesanne lahendamiseks arvestades probleemi konteksti. Võrdlusanalüüsi kriteeriumid on järgmised ning põhinevad sellel artiklil [6]:

- Teegi funktsionaalsus. Mis funktsionaalsust teek pakkub ülesanne lahendamiseks.
- Teegi populaarsus. Kui aktiivselt tehnoloogiat arendatakse ning kui palju inimesi kasutab tehnoloogiat. See on tähtis sellepärast, et populaarsemate tehnoloogiate jaoks eksisteerib rohkem juhiseid ja teist tüübi infot internetis. Populaarsuse mõõtmiseks on kasutatud tärnid teekide repositooriumites Github platvormil.
- Teegi dokumentatsiooni olemasolu ja selle täpsus. Sellest sõltub seda, kui mugav ja effektiivne on teegi kasutamise protsess. Juhul kui teek on halvasti dokumenteeritud, siis selle kasutamine on aeglasem kuna tuleb teha lisauuringut, et teada kuidas teek töötab.
- Teegi kasutamise keerukus. Samuti mõjutab seda, kui effektiivne on teegi kasutus. Juhul kui teegi on liiga raske kasutada, siis teegi kasutuselevõtmine vajab lisauuringut. Teegi kasutamise keerukuse näidiseks on keeruline konfimine. Juhul kui teegi konfimiseks ei ole vaja palju lisainfot teada ja konfimine toimub mitu koodiridade kirjutamisega, siis selle teegi kasutamise keerukus on väiksem. Juhul kui teegi konfimiseks on vaja kirjutada mitu kümenid koodiridu ja nende korrektne kirjutamine vajab häid tehnoloogia teadmist, siis tehnoloogia

kasutamise keerukus on suurem. Selleks et aru saada kas tehnoloogiat on lihtne või keeruline kasutada töö autor proovib igat tehnoloogiat kasutada ise.

3.3.1 API versioonide võrdlemise teek

Peatükkis 2 oli mainitud, et Spring raamistiku uue versiooni ilmumisega raamistiku API muudetud ja eemaldatud osad ei ole Spring arendustiimi poolt täpselt dokumenteeritud. Sellepärast arendatav rakendus peab olema võimeline koguma seda infot ise. Uuringu jooksul oli leitud hulk väliseid teeke, mis võimaldavad võrrelda sama API kahte erinevat versiooni. Sellised tööriistad lasevad koguda infot muutmata jäänud, uutest, muudetud ja eemaldatud API osadest.

Uuringu jooksul oli leitud 5 eelmises lõigus mainitud tüübi teegi. Nendest teekidest 3 ei ole enam toetatud ehk nende arendus peatus liiga ammu ja nad ei sisalda kogu vajalikku funktsionaalsust püstitatud ülesanne lahendamiseks. Peamine puudujak kõikides vanades teekides on see, et nad ei kogu informatsiooni anotatsioonidest, millised Spring raamistikus on olemas ning on osa raamistiku baasfunktsionaalsusest. Sellepärast oli otsustatud neid teeke mitte kasutada. Teegid, mis ei ole toetatud on *Clirr*, mille arendus peatus aastal 2005 [7], *JDiff*, mille arendus peatus aastal 2013 [8] ja *Java Api Compliance Checker*, mida ei arendata aastast 2021 [9]. Jäänud 2 teegid on uuemad, sellepärast võrdlusanalüüs toimub ainult nende kahe vahel.

Japicmp on avatud lähtekoodiga kättesaadav CLI tööriistana ja eraldi teegina, mis võimaldab võrrelda sisendina antud ühe teegi erinevate versioonidega jar faile. Teek sisaldab kogu funktsionaalsust selles peatükis mainitud probleemi lahendamiseks ja on paindlikult konfitav. Teegi arendatakse tänini ja teegi githubil on 656 tärni [10]. Teegi dokumentatsioon on ebatäielik. On olemas javadoc API dokumentatsioon ja lihtne sissejuhatav juhend, aga sellist ei piisa et oleks lihtne arusaada, kuidas antud teegi kasutada. Teek ei ole suur ja sellepärast sellest on võimalik aru saada lähtekoodi lugedes mõistliku aja jooksul. Teegi on lihtne kasutada. [11]

Revapi on avatud lähtekoodiga CLI tööriist ja teek, mis oli loodud eesmärgiga võrrelda API versioonide muutmist, sõltumata tehnoloogiast millel API põhineb. Teegi arendus peatus aastal 2022 ja teegi githubil on 171 tärni [12]. Võrreldes japicmp ja teiste sarnaste teekidega, revapi on disainitud niimoodi, et seda oleks võimalik kasutada suvalisel tehnoloogial põhineva API skaneerimiseks, mitte ainult Javal. Lisaks lähtekoodi

skaneerimisele teek on võimeline skaneerida konfiguratsiooni faile, millised võivad ka skaneeritava teegi lähtekoodis ilmuda. Teegi jaoks on olemas ka juhend, kuidas seda laiendada et see võimaldaks skaneerida teisel tehnoloogial baseeruvaid teeke. Teegi dokumentatsioon on ebatäielik. On olemas javadoc dokumentatsioon ja sissejuhatav juhend. Võrreldes japicmpga, revapi sissejuhatava dokumentatsiooni sisu oli mõnedes kohtades väär ning dokumentatsiooni järgi CLI tööriista kasutamine ei õnnestu. Töö autor pidi ise uurima miks see ei töötanud ning uurimise lõpuks CLI tööriist hakkas töötama. Töö kirjutamise hetkel tööriist on võimeline võrrelda Java teeke, JSON faile ja YAML faile. [13]

Tabel 1. API versioonide võrdlemise teekide võrdlusanalüüsi tulemus

Teegi nimi	**Funktsionaalsus**	**Populaarsus**	**Dokumentatsioon**	**Kasutamise keerukus**
Japicmp	Laseb skaneerida Java teeke [11]	656 tärni githubil. Teegi arendatakse tänini [10]	Sissejuhatavad juhendused + javadoc [11]	Lihtne
Revapi	Laseb skaneerida Java teeke ning JSON ja YAML konfiguratsioonifaile [13]	171 tärnig githubil. Aastast 2022 teegi enam ei arendata [12]	Sissejuhatavad juhendused + javadoc. Mõnedes kohtades dokumentatsioon on aegunud [13]	Lihtne

Tabel 1 näitab API versioonide võrdlemise teekide võrdlusanalüüsi tulemust. Oli otsustatud kasutada japicmp teegi. Püstitatud probleem on aktuaalne ainult Java tehnoloogiatel baseeruvate lahenduste jaoks, ning sellepärast see teek hästi sobib selle ülesanne lahendamiseks. Teegi aktiivselt arendatakse töö kirjutmise hetkel. Kui reliisitakse uued Java programmeerimiskeele versioonid, siis nad saavad toetust japicmp teegi poolt. Revapi teek ei ole arenduses juba 1 aasta. Teiste aspektide mõttes teegid on

võrdsad. Mõlemal teegil on ebatäielik dokumentatsioon, mõlemaid teeke on sarnaselt lihtne kasutada.

Revapi teegi oleks mõistlik kasutada juhul, kui oleks eesmärk luua universaalsemat tööriista. Juhul kui probleemi skoobis oleksid ka teistel tehnoloogiatel baseeruvate teekide analüüs, siis teek sobiks probleemi lahendamiseks paremini. Lisaks antud stsenaariumil teegi kasutamine vajaks rohkem arendust. Teek toetab ainult Java, JSON ja YAML skaneerimist ning teiste tehnoloogiate jaoks tuleks iseseisvalt arendada laiendusi.

3.3.2 Programmeerimiskeel

Kuna japicmp teek on kättesaadav ainult Java teegi vormis on otsustatud rakenduse arendamiseks kasutada Java programmeerimiskeelt. Java on üks kõige populaarsematest programeerimiskeeltes maailmas ja töö autoril on olemas töökogemus antud programmeerimiskeelega. Sellepärast ei ole suureks puuduseks, et tekkis piirang programmeerimiskeele valikul.

Java reliiside versioonid jagavad kaheks: featuur- ja LTS-reliisideks. Featuur reliisid on toetatud 6 kuu jooksul ja pakkuvad uued programmeerimiskeele featuurid. LTS-reliisid pakkuvad ainult vigade, turva ja jõudluse parandusi ja neid reliisi toetatakse vähemalt 8 aastat [14]. Kuna rakenduse kirjutamise jaoks ei ole vaja kasutada uued featuurid, mis pakkuvad featuur-reliisid, oli otsustatud kasutada viimast Java LTS versiooni. Töö kirjutamise hetkel viimane Java LTS versioon on 21 ning seda kasutatakse rakenduse arendamiseks [15].

Kuna projekti programmeerimiskeeleks on Java, edasi otsitakse teeke ainult selle programmeerimiskeele jaoks.

3.3.3 Ehitamistööriist

Rakenduse arenduse jaoks on plaanis kasutada väliseid teeke, mis asuvad välisveebis. Välissõltuvuste haldamine on kompleksne ning selle jaoks tavaliselt kasutatakse eraldist tööriistat, mida nimetatakse ehitamistööriistaks ehk ehitamissüsteemiks [16].

Kõige populaarsemad Java ehitamistööriistad on Gradle ja Maven. JetBrains tarkvaraarenduse ökosüsteemi seisundi 2023 aasta uuringu järgi Gradle tööriista kasutab 46% uuringu vastajatelt ning Maven tööriista kasutab 74% uuringu vaastajatelt [17]. Plaanitava rakenduse kontekstis ei ole plaanis teha keerulist projekti ehitamise struktuuri.

Sellepärast peamised omadused mille järgi ehitamistööriist on valitud on seotud tööriista kasutamisemugavusega. Gradle on kahe korda kiirem kui Maven peaaegu kõikide ehitamisülesannete täitmisel [18]. Gradle ehitamisskriptide kirjutamiseks on kasutatud Gradle ehitamise keel mis baseerub Groovy DSL tehnoloogial. Maven ehitamisskriptide kirjutamiseks on kasutatud XML. Gradle ehitamisskriptid tavaliselt on kompaktsemad ja paremini loetavamad kui Maven ehitamisskriptid [19].

Ülal toodud põhjuste tõttu oli otsustatud kasutada Gradle ehitamistööriista arendatava rakenduse ehitamise protsessi lihtsustamiseks.

3.3.4 Java lähtekoodi skaneerimise teek

Selleks et leida japicmp teegiga kogutud info alusel ebaühilduvaid koodiosad skannitavas teegis on vaja kasutada eraldist tööriistad, mis saab Java programmi struktuuri teisendada Java objektide struktuuriks. Spring API kasutatud osade skannitava teegi lähtekoodist leidmiseks on vaja teha skannitava koodi lisaanalüüsi. Koodi analüüs on kirjeldatud detailsemalt peatükis 4.4. Uuringu jooksul töö autor leidis mitu teeke, mis lasevad genereerida abstraktset süntaks puud Java objektide kujul. Abstraktne süntaks puu ehk AST on struktuur, mida on võimalik kasutada suvalisel programmeerimis keelel kirutatud programmi struktuuri esitamiseks, muutmiseks ja uue koodi genereerimiseks [20]. Kuna antud probleemi lahendamiseks eksisteerib palju erinevaid teekie, võrdlusanalüüsis vaadetakse ainult neid teeke, mis on kõige populaarsemad. Kõik võrdlusanalüüsis mainitud teegid töö kirjutamise hetkel on aktiivses arenduses.

JavaParser on kõige populaarsem teek Java koodi parsimiseks ja genereerimiseks. Teek sisaldab piisavalt funktsionaalsust ülesanne lahendamiseks. Teegi github repositooriumis on 4987 tärni töö kirjutamise hetkel [21]. Teek on väga hästi dokumenteeritud. Dokumentatsioon on kirjutatud raamatu kujul ja on väga detailne ning lihtsasti arusaadav. On olemas javadoc dokumentatsioon. Teegi kasutamine on lihtne ja intuitiivne. [22]

Spoon on sama eesmärgiga loodud tööriist, nagu Javaparser. Teek on kasutatud Java koodi parsimiseks, muutmiseks ja genereerimiseks. Teegi funktsionaalsus on väga lai. Võrreldes JavaParser teegiga, Spoon sisaldab rohkem viisi abstraktse süntaks puu elementide navigeerimiseks. Teegi githubil on 1594 tärni töö kirjutamise hetkel [23]. Teek on detailselt dokumenteeritud, on olemas javadoc dokumentatsioon. Teegi on lihtne ja intuitiivne kasutada. [24]

ANTLR on tööriist parserite genereerimiseks. Seda kasutatakse programeerimiskeelte, raamistikute ja teiste tööriistade loomise jaoks. ANTLR funktsionaalsuse komplekt on mastaapsem ja komplekssem võrreldes kahe eelmiste teekidega, kuna seda tööriista kasutatakse programmeerimiskeelte loomiseks. Teegi githubil on 15160 tärni töö kirjutamise hetkel [25]. Teegi dokumentatsioon eksisteerib detailses raamatu eksemplaaris ja kompaktsemas eksemplaaris teegi githubil. On olemas javadoc dokumentatsioon. Java koodi parsimise jaoks ANTLR tööriista kasutamine on keeruline, võrreldes teiste analüüsis osalevate tööriistadega. [26]

Eclipse JDT Core on tööriistade komplekt Java programmide arenduseks Eclipse IDEs, mida on võimalik iseseisva komponendina kasutada sõltumata IDEst. Püstitatud probleemi lahendamiseks Eclpise JDT Core komponent sisaldab funktsionaalsust Java lähtekoodi AST mudeli genereermiseks. Komponendi githubil on 91 tärni töö kirjutamise hetkel [27]. AST mudeli genereerimise funktsionaalsuse jaoks on olemas ainult javadoc dokumentatsioon. Sellepärast teegi funktsionaalsust ei ole võimalik kergesti õppida. [28]

Tabel 2 näitab Java lähtekoodi skaneerimise teekide võrdlusanalüüsi tulemust. ANTLR teek ei ole sobilik valik, kuna selle kasutamine on liiga keeruline võrreldes teiste lahendustega ning selle lai funktsionaalsus ei ole probleemi lahendamiseks vajalik. ANTLR tööriist sobiks juhul, kui oleks eesmärk luua universaalsemat tööriista, mis võiks analüüsida teistelt tehnoloogiatel põhinevaid lahendusi. Seda oleks võimalik kasutada kombinatsioonis koos Revapi teegiga, mis oli mainitud allpetükis 3.3.1.

Eclipse JDT Core ei ole populaarne tööriist ning selle dokumentatsioon ei ole mugav. See tööriist võiks olla sobivaks valikuks juhul, kui töö autor kasutaks Eclipse IDEd oma lahenduse arenduseks. Sellepärast, et tööriist võrreldes teistega ei ole väga populaarne ja on halvemini dokumenteeritud ning töö autori töökohal kasutatakse Intellij IDEA IDEd, töö autor ei kasuta seda lahendust.

Võrdlemiseks jäid kaks teeke, Spoon ja JavaParser. Java lähtekoodi AST mudeli parsimise ja uurimise funktsionaalsust Spoon teegil on rohkem ja selles mõttes see on parem. JavaParser teek on paremini dokumenteeritud ja on populaarsem kui Spoon teek. Seetõttu võrreldes teiste teekidega töö autorile oli kõige mugavam kasutada JavaParser teegi ning see oli valituda ülesanne lahendamiseks. Teegi funktsionaalsusest piisab püstitatud probleemi lahendamiseks. Teegist on lihtne aru saada, sest selle jaoks on

olemas hea dokumentatsioon. Kui tekkisid triviaalsed probleemid teegi kasutamisel, veebis on lihtne leida probleemide lahendust, kuna JavaParser teek on populaarne.

Tabel 2. Java lähtekoodi skaneerimise teekide võrdlusanalüüsi tulemus

Teegi nimi	**Funktsionaalsus**	**Populaarsus**	**Dokumentatsioon**	**Kasutamise keerukus**
JavaParser	Teek laseb parsida Java koodi [22]	4987 tärni githubil [21]	Väga detailne ja lihtsasti arusaadav + javadoc [22]	Lihtne
Spoon	Teek laseb parsida Java koodi [24]	1594 tärni githubil [23]	Detailne dokumentatsioon + javadoc [24]	Lihtne
ANTRL	Teek laseb parsida paljude programeerimiskeelte koodi [26]	15160 tärni githubil [25]	Detailne dokumentatsioon + javadoc [26]	Keeruline
Eclipse JDT Core	Teek laseb parsida Java koodi [28]	91 tärni githubil [27]	Ainult javadoc [28]	Keskmine

3.3.5 HTML lehtede genereerimise teek

Aruanne genereerimine toimub japicmp ja JavaParser tööristadega kogutud info alusel. Uuringu jooksul tööautor leidis mitu tööriistaid, mis lasevad genereerida HTML ja PDF faile. Oli otsustatud analüüsida ainult HTML genereerimise tööriistaid, sest HTML lehtede loomine on paindlikum ja mugavam, kui PDF dokumentide loomine. Teiseks põhjuseks on see, et töö autoril on rohkem kogemust HTML failide programmilise loomisega. Teegid mis lasevad luua HTML lehte kasutades andmeid mõni programmeerimiskeele objektidest on nimetatud malli mootoriteks. Eksisteerib palju Javal põhinevaid malli mootorite teeke. Sellepärast võrdlusanalüüsis käsitletakse ainult neid teeke, milliseid tänini aktiivselt arendatakse. On võrreldatud 3 malli mootoreid.

Thymeleaf on kaasaegne malli mootor. Teek sisaldab kogu vajalikku funktsionaalsust HTML lehtede genereerimiseks kasutades andmeid Java objektidest. Teegi githubil on 2650 tärni töö kirjutamise hetkel [29]. Teegi jaoks on olemas detailne ja lihtsasti arusaadav dokumentatsioon. Teegi on lihtne kasutada. [30]

Antud analüüsis ainult Thymeleaf malli mootoriga töö autor sai töökogemust. Töö autori töökoha meeskonna liikmed on ka tuttavad selle tehnoloogiaga. See annab lisa eelist sellele tehnoloogiale võrreldes teiste tehnoloogiatega antud analüüsis.

Freemarker on sarnane tööriist nagu Thymeleaf. Funktsionaalsuse mõttes Freemarker on võimeline teha samuti palju, kui Thymeleaf. FreeMarker githubil on 896 tärni töö kirjutamise hetkel [31]. Teegi dokumentatsioon on detailne ja lihtsasti arusaadav. Teegi on lihtne kasutada. [32]

Pebble on sarnane tööriist nagu Freemarker ja Thymeleaf. Võrreldes eelmiste kahe lahendustega Pebble teegi funktsionaalsus on kõige minimalistlikum ja samal ajal sobib püstitatud probleemi lahendamiseks. Pebble githubil on 1038 tärni töö kirjutamise hetkel [33]. Teegi dokumentatsioon on detailne ja kõige arusaadavam võrreldes eelmiste lahendustega. Teegi on kõige lihtsam kasutada võrreldes eelmiste lahendustega. [34]

Tabel 3 näitab HTML lehtede genereerimise teekide võrdlusanalüüsi tulemust. Oli otsustatud kasutada Thymeleaf teegi. See on kõige populaarsem teek antud kolmest lahendustest, aga teistel parameetritel Pebble teek on parem. Thymeleaf oli valitud kuna töö autoril ja tema kolleegidel on olemas kogemus antud tehnoloogiaga ning sellepärast selle kasutamine antud kontekstis on kõige lihtsam ja effektiivsem.

Lisaks oli otustatud visuaalse aruanne stiliseerimiseks kasutada Bootstrap teegi. Bootstrap on teek, mis annab hulk steliseeritud HTML komponente nagu tabelid ja vormid [35]. Sellel tehnoloogial on väga palju alternatiive, aga antud töös neid ei käsitletata. Bootstrap oli valitud, kuna töö autoril on olemas kogemus selle tehnoloogiaga.

Tabel 3. HTML lehtede genereerimise teekide võrdlusanalüüsi tulemus

Teegi nimi	**Funktsionaalsus**	**Populaarsus**	**Dokumentatsioon**	**Kasutamise keerukus**
Thymeleaf	Laseb genereerida HTML lehte kasutades Java objekte [30]	2650 tärni githubil [29]	Detailne ja lihtsasti arusaadav dokumentatsioon [30]	Väga lihtne. Töö autoril on olemas töö kogemus antud teegiga
Freemarker	Laseb genereerida HTML lehte kasutades Java objekte [32]	896 tärni githubil [31]	Detailne ja lihtsasti arusaadav dokumentatsioon [32]	Lihtne
Pebble	Laseb genereerida HTML lehte kasutades Java objekte. Teegi funktsionaalsus on väga minimalistlik [34]	1038 tärni githubil [33]	Detailne ja väga lihtsasti arusaadav dokumentatsioon [34]	Väga lihtne

3.3.6 CLI teek

Lähtudes rakenduse funktsionaalsetest nõuetest rakenduse jaoks tuleb arendada CLI liidest. Uuringu jooksul töö autor leidis mitu tehnoloogiat CLI liidese loomiseks. CLI liidest on võimalik implementeerida kasutades ainult Java API [36]. Selle lähenemise eeliseks on see, et see on väga paindlik. Teiseks eeliseks on see, et ei tekki lisasõltuvust teisest tehnoloogiast rakenduse jaoks. Puuduseks on see, et CLI rakenduse loomine kasutades ainult Java API võtab palju aega. Plaanis ei ole luua kompleeksset CLI liidest.

Sellepärast on mõistlik leida teist tööriista, mis võimaldab luua CLI liidest kiiremini ja lihtsamini. Uuringu jooksul oli leitud teegid, mis võimaldavad luua CLI liidest Java baasil. Java CLI rakenduste loomiseks eksisteerib palju teeke. Sellepärast oli otsustatud antud võrdlusanalüüsis käsitleda ainult 4 kõige populaarsemaid teeke, milliseid arendatakse töö kirjutamise hetkel.

Picocli on kõige populaarsem Java programmeerimiskeelel põhinev teek CLI rakenduste loomiseks. Teek sisaldab kogu vajalikku funktsionaalsust CLI rakenduse loomiseks. Teegi jaoks on olemas ametlik integratsioon Spring Boot raamistikuga ja mitu teiste populaarsemate tehnoloogiatega. Teegi githubil on 4499 tärni töö kirjutamise hetkel [37]. Teek on detailselt dokumenteeritud ja selle jaoks on olemas javadoc dokumentatsioon. Teegi on lihtne ja intuitiivne kasutada. [38]

JCommander on sama tüübi teek nagu Picocli. Teegi abil on võimalik CLI rakendusi luua. Teegi githubil on 1891 tärni töö kirjutamise hetkel [39]. Teek on detailselt dokumenteeritud ja selle jaoks on olemas javadoc dokumentatsioon. Teegi on lihtne ja intuitiivne kasutada. [40]

Spring Shell on Spring raamistiku osa, mis võimaldab luua CLI rakendusi. Võrreldes kahe eelmiste teekidega, see teek tuleb kaasa täisjõulise CLI keskkonnaga, millisel on olemas oma featuurid nagu käsude lugu, vaikimisi sisseehitatud käsud ja otsene integratsioon Spring raamistikuga. Teegi githubil on 684 tärni töö kirjutamise hetkel [41]. Teek on detailselt dokumenteeritud ja selle jaoks on olemas javadoc dokumentatsioon. Teegi on lihtne ja intuitiivne kasutada. [42]

Apache commons CLI on sarnane teek nagu Picocli või JCommander. Võrreldes kõikide eelmiste teekidega, teegi kasutamine on programmiline. Kõiki teisi teeki antud analüüsis on võimalik kasutada Java annotatsioonidega, mis on mugavam ja loetavam. Teegi githubil on 312 tärni töö kirjutamise hetkel [43]. Teegi jaoks on olemas sissejuhatav dokumentatsioon ja javadoc dokumentatsioon. Teegi kasutamine ei ole lihtne, sest ei ole võimalustada kasutada teegi annotatsioonidega [44]

Tabelid 4 ja 5 näitavad CLI teekide võrdlusanalüüsi tulemust. Oli otsustatud kasutada Picocli teegi. Spring Shell teegiga tehtud kasutajaliides on keerulisem, kui rakendus seda nõuab. Rakenduse kasutamiseks ei ole vaja palju kasutajaliidese funktsionaalsust, sellepärast kogu Spring Shelli eraldine keskkond on liigne antud rakenduse jaoks. Apache

commons CLI on ebamugavam ja vähem populaarne kui picocli teek. JCommander on samuti mugav ja populaarne nagu Picocli, aga selle jaoks ei ole ametlikke integratsioone teiste tehnoloogiatega.

Tabel 4. CLI teekide võrdlusanalüüsi tulemus (1)

Teegi nimi	**Funktsionaalsus**	**Populaarsus**	**Dokumentatsioon**	**Kasutamise keerukus**
Picocli	Teek laseb ehitada CLI liidest Java programmide jaoks. On olemas võimalus kasutada teegi Java annotatsioonidega ja programmaatiliselt. Integratsioon Spring raamistikuga [38]	4499 tärni githubil [37]	Detailne dokumentatsioon + javadoc [38]	Lihtne ja intuitiivne
Jcommander	Teek laseb ehitada CLI liidest Java programmide jaoks. On olemas võimalus kasutada teegi Java annotatsioonidega ja programmaatiliselt [40]	1891 tärni githubil [39]	Detailne dokumentatsioon + javadoc [40]	Lihtne ja intuitiivne

Tabel 5. CLI teekide võrdlusanalüüsi tulemus (2)

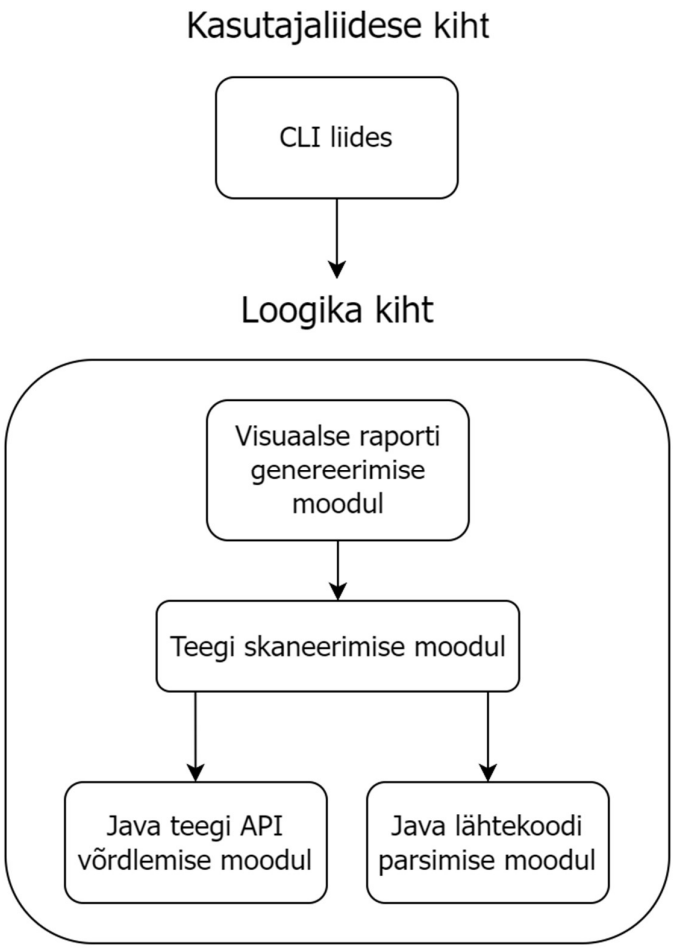
Spring Shell	Teek laseb ehitada keerulist CLI liidest Java programmide jaoks + integrastioon Spring raamistikuga [42]	684 tärni githubil [41]	Detailne dokumentatsioon + javadoc [42]	Lihtne ja intuitiivne
Apache commons CLI	Teek laseb ehitada CLI liidest Java programmide jaoks. Puudub võimalust kasutada teegi Java annotatsioonidega [44]	312 tärni githubil [43]	Sissejuhatav dokumentatsioon + javadoc [44]	Keeruline

3.4 Plaanitava lahenduse projekteerimine

Lähtudes enne defineeritud funktsionaalsetest ja mittefunktsionaalsetest nõuetest ning valitud tehnoloogiatest, püstitatud probleemi lahendamiseks oli projekteeritud rakenduse arhitektuuri prototüüp (Joonis 1).

Arhitektuur oli tehtud lähtudes Clean Code raamatus mainitud printsiibist *Separation of Concerns*, mis on käsitletud raamatu peatükkis 11 [45]. Rakendused, mis järgivad seda printsiibi on lihtsasti testitavad ja moduleeritavad. Need omadused on kasulikud disainitava rakenduse jaoks, sest lähtudes mittefunktsionaalsetest nõuetest rakendus peab olema lihtsasti testitav ja laiendatav ning selle struktuur peab olema kergesti arusaadav.

Rakendus koosneb kahest peamisest kihist: kasutajaliidese kiht ja loogika kiht. Kasutajaliidese kihi moodulid sõltuvad loogika kihi moodulitest. Esialgu kasutajaliidese kiht koosneb ainult ühest moodulist mis vastab CLI kasutajaliidese eest. See jagamine on tehtud seetõttu, et on võimalik et tulevikus esinevad uued kasutajaliidesed, mis antud arhitektuuri kontekstis võivad taaskasutada loogika kihi funktsionaalsust. Uue kasutajaliidese näide on API liides, mis on vajalik juhul kui otsustatakse teha sellel raakendusel baseeruv veebirakendust. CLI liides mooduli implementeerimiseks on kasutatud Picocli teek.



Joonis 1. Rakenduse arhitektuuri prototüüb

Loogika kihi esimene moodul on visuaalse raporti genereerimise moodul, mis omakorda kasutab teegi skaneerimise moodulit. See moodul on vastutav visuaalse raportii koostamise eest kasutades teegi skaneerimise moodulist saadud informatsiooni ebaühilduvatest koodiridadest. Raporti genereerimise moodulis on kasutatud Thymeleaf malli mootor.

Teegi skaneerimise moodul sõltub Spring raamistiku API võrdlemise ja Java lähtekoodi parsimise moodulitest. See moodul kasutab kahest teistest moodulitest korjatud infot ja otsib skaneeritavast teegist Spring APIst eemaldatud ja muudetud osad ning säilitab funktsionaalnõuetes mainitud infot iga ebaühilduva koodirea kohta.

Java teegi API võrdlemise moodul võrdleb kahe Java teegi API versiooni ning korjab infot eemaldatud ja muudetud teegi API osadest uuemas teegi API versioonis. Esialgu moodulis toimub ainult põhiliste Spring teekide API võrdlemine versioonide 5 ja 6 vahel.

Tulevikus on võimalik seda moodulit laiendada ja lisada skaneerimist teiste Spring raamistiku versioonide vahel. Selles moodulis on kasutatud japicmp teek.

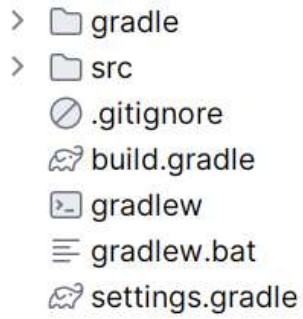
Java lähtekoodi parsimise moodul laseb parsida Java lähtekoodi Java objektide struktuuriks, mida on võimalik Java programmiga mugavalt analüüsida. Selles moodulis on kasutatud JavaParser teek.

4 Rakenduse arendus

Antud peatükis on käsitletud rakenduse arendus, mis põhineb peatükkis 3 tehtud analüüsil. Antud peatükis on kirjeldatud rakenduse struktuur ja tehnilised lahendused iga rakenduse komponendi jaoks. Peatüki lõpus on kirjeldatud rakenduse testimise protsess ja on tehtud kokkuvõtte saadud tulemuste kohta rakenduse arenduse jooksul.

4.1 Projekti struktuur

Projekti struktuur kujunes peatükkis 3.4 välja toodud disaini järgi ja rakenduse arenduse jooksul. Projekti juurstruktuur on näidatud joonisel 2:



Joonis 2. Projekti juurstruktuur

Kaust gradle sisaldab endas Gradle wrapperi konfiguratsiooni ning koodi Gradle ehitamistööriista laadimiseks. Gradlew on Gradle wrapperi shell skript ja gradlew.bat on Windows batch Gradle wrapperi skript. Gradle wrapper on skript, mis laseb käivitada gradle kaustas asuva konfiguratsiooni järgi valitud Gradle ehitamistööriista versiooni. See annab võimalust vajadusel mugavalt muuta projektis kasutatud Gradle ehitamistööriista versiooni [46].

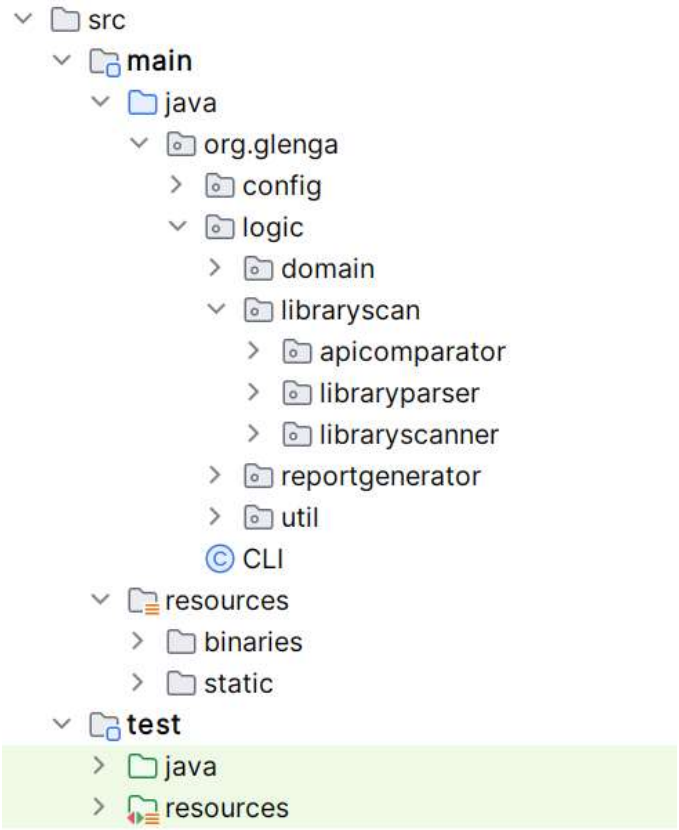
Kaust src sisaldab rakenduse Java lähtekoodi ja unit teste.

Fail build.gradle on Gradle tööriista jaoks vajalik ehitamisskript, mille järgi Gradle projekt on ehitatud. Antud projekti kontekstis skript on kasutatud projekti sõltuvuste haldamiseks. [47].

Projektis on kasutatud versioonide halduse süteem git. Failis .gitignore on loendatud need failid, mis git versioonide halduse süsteem ei jälgi [48].

Fail settings.gradle sisaldab Gradle ehitamistööriista konfiguratsiooni [49].

Projekti Java koodi struktuur on näidatud joonisel 3.



Joonis 3. Projekti Java lähtekoodi struktuur

Main kaust sisaldab endas java ja resources kaustad. Kaust java sisaldab java lähtekoodi. Kaustad org ja glenga formeerivad projekti ainuket Java packaget. Package org.glenga oli genereeritud projekti loomisel Intellij IDEA IDEs. Java package on nimede ruum, mis laseb organeseerida hulk Java klasse ja liideseid [50].

Config kaust sisaldab klasse, mis vastutavad rakenduse konfigureerimise eest.

Logic kaust sisaldab klasse ja liideseid, mis vastutavad peatükis 3.4 mainitud logika kihi implementeerimise eest. Kaust domain sisaldab domeen objekte ja eraldiseid klasse domeen objektide loomiseks. Kaust libraryscan sisaldab kogu loogikat teegi skaneerimise jaoks. Siin asub kood, mis implementeerib sisendteegi lähtekoodi skaneerimist, Spring

raamistiku API võrdlemist ja vananenud Spring API kasutamist otsingut sisendteegis. Selline grupeerimine on põhjendatud sellega, et klassid mis realiseeruvad need kolm osa on tihedalt seotud, sest nad kõik kasutavad japicmp ja JavaParser teekide klasse. Kaust reportgenerator sisaldab koodi visuaalse aruanne genereerimiseks. Kaust util sisaldab abimeetodeid mis võib igaüks rakenduse moodul kasutada.

CLI klass on rakenduse põhiklass, millest alustab rakenduse töö. CLI klaas on vastutav CLI kasutajaliidese funktsionaalsuse tagamise eest ja rakenduse konfigureerimise eest.

Resources kaustas asuvad rakenduse ressursid. Kaust binaries on ette nähtud kõikide vajalikke kompileeritud java teekide jaoks, mis on vajalikud JarTypeSolver sümbolite lahendajate konfigureerimiseks (peatükk 4.3) ja japicmp teegi sisendiks (peatükk 4.2). Töö kirjutamise hetkel selles kaustas asuvad Spring raamistiku osad. Static kaustats asuvad staatilised ressursid, mis on vajalikud HTML raportite genereerimiseks. Nendeks ressursideks on JavaScript ja CSS failid ning mall, mida kasutab Thymeleaf teek HTML raporti genereerimiseks. Selles kataloogis asub kompileeritud Bootstrap teek.

Test kataloogis asuvad rakenduse unit testid.

4.2 Info kogumine muudetud ja eemaldatud Spring raamistiku API osadest

Antud ülesanne lahendamiseks oli kasutatud japicmp teek. Teek sisaldab klassi JarArchiveComparator. Klass sisaldab meetodi compare, mis võtab sama API kahe erineva versiooni kompileeritud koodiga jar faile ja võrdleb neid. Meetod tagastab loendi japicmp teegi klassi JApiClass objektidest. JApiClass klass kajastab informatsiooni skannitava teegi ühe API klassi kohta. Klass näitab API klassi staatust võrreldes eelmise teegi versiooniga. Võimalikke API klassi staatuseid on neli: muudetud, uus, eemaldatud ja muutmata. JApiClass klass sisaldab palju erinevat infot analüüsitavatest klassist. Antud töö kontekstis on tähtis teada et JApiClass klass sisaldab kõiki klassi konstruktoreid, liideseid milliseid klass impelementeerib ja klassi meetodeid. Klassid mis kajastavad neid keele konstruktsioone võivad omada samad neli staatust, mis näitab JApiClass klass. API versioonide võrdlemise koodi näitab lisa 2. [51]

Meetod getModifiedAndRemovedJApiObjectsWithChangeStatus filtreerib japicmp teegi objekte mis omavad muudatuse staatust. Meetod tagastab neid objekte, mis on staatusega

muudetud või eemaldatud, sest ainult nende API osade kasutamine teeb sisendteegi ebaühilduvaks uuema Spring raamistiku versiooniga. Veel see meetod on taaskasutatud sisendteegi lähtekoodist vananenud Spring API osade otsimisel, millest on kirjutatud peatükkis 4.4.

Japicmp teegi konfiguratsioonist on aktiveeritud kaks sätted: setIgnoreMissingClasses ja setOutputOnlyModifications. Säte setIgnoreMissingClasses aktiveerimise juhul, japicmp ignoreerib neid teeke klasse skaneerimisel, mis implementeerivad liideseid ja laiendavad klasse välisteekidest, mis skaneerimise hetkel ei ole määratud Java classpathis. Antud juhul ei ole võimalik kontrollida binaarset ühilduvust klassidel, mis implementeerivad liidest või laiendavad klassi sõltuvusena kasutatud teegist [52]. Selleks et automaatselt lisada kõiki Spring teekide sõltuvusi on tarvis arendada uut Java moodulit, mis antud töö raames ei ole tehtud. Kuna binaarne ühilduvus ei ole tähtis antud rakenduse kontekstis mis on tõendatud peatükis 4.4, selle säte akriveerimine on aktsepteeritav ja laseb kokku hoida arenduse aega. Säte setOutputOnlyModifications aktiveerimisel JarArchiveComparator.compare meetodi kasutamisel on tagastatud ainult need klassid, mille vähemalt ühe elemendi muudatuse staatus on eemaldatud, muudetud või uus [52].

4.3 Sisendteegi lähtekoodi skaneerimine

Antud ülesanne lahendamiseks oli kastutatud JavaParser teek. Teek annab võimalust teisendada sisendina antud lähtekoodi AST mudeliks. Peatükis 4.4 kirjeldatud funktsionaalsuse tagamiseks oli otsustada kasutada JavaParser teegi sümbolite lahendamise (*symbol solving*) funktsionaalsust. Antud kontekstis sümbolid on kõik nimed, mis ilmuvad parsitavas lähtekoodis. Vaikimisi kui teek parsib Java lähtekoodi, see ei korja informatsiooni sellest, mida need nimed kajastavad. Ilma lisa analüüsi pole teada, kas mingi sümbol on muutuja, klassi staatiline väli või staatiline meetod. Peatükis 4.4 funktsionaalsuse tagamiseks tuleb uurida mis Java tüübi nimed kajastavad. Selleks on võimalik kasutatda JavaParser sümbolite lahendamist. [22]

Sümbolite lahendamise töö tagamiseks on vaja konfigureerida sümbolite lahendajat (*symbol resolver*) ning omistada seda JavaParser globaalse parseri konfiguratsioonis. Sümbolite lahendajad analüüsivad Java klasse ja neid eksisteerib mitu tüübi. Vajaliku funktsionaalsuse implementeerimiseks oli otsustatud kasutada kaks sümbolite lahendajate tüübi: ReflectionTypeSolver, JarTypeSolver ja CombinedTypeSolver.

ReflectionTypeSolver analüüsib Java API klasse ning oma konfigureerimiseks ei nõua mingit lisa sisendparameetri. JarTypeSolver analüüsib kompileeritud Java klasse ja oma loomiseks vajab sisendina antud jar faili kompileeritud Java koodiga. CombinedTypeSolver on sümbolite lahendaja, mis koosneb mitmetest sümbolite lahendajatest. Seda kasutakse sellepärast, et JavaParser konfiguratsioon saab hoida ainult üht sümbolite lahendajat. Antud töö kontekstis CombinedTypeSolver hoiab üht ReflectionTypeSolver tüübi sümbolite lahendajat ja kõiki JarTypeSolver tüübi sümbolite lahendajaid. [22]

JarTypeSolver tüübi sümbolite lahendajad on konfigureeritud Spring versiooni 5 teekide klasside jaoks, nende teekide sõltuvuste klasside jaoks ja sisendteegi klasside jaoks. Spring raamistikuga teekide ja nende sõltuvuste konfiguratsioonid on vajalikud, sest skannitavas sisendteegis võivad esineda sümbolid mis on seotud Spring raamistiku teekide ja nende sõltuvuste klassidega. Konfiguratsioon sisendteegi jaoks on vajalik, sest Spring raamistiku teekide API kasutamisel on võimalik, et Spring API osadesse sisendina on antud sümbolid mis on seotud sisendteegi klassidega. See on võimalik näiteks kui on kasutatud suvaline meetod Spring teegi APIst mis võtab vastu sisendparameetrina java.lang.Object klassi objekti ja sisendparametrina on antud mingi sisendteegi klass, mis on tuletatud sellest klassist. Sellepärast arendatav programm vajab sisendparametrina teed jar failile sisendteegi kompileeritud koodiga.

Sisendteegi koodi parsimiseks on tehtud meetod, mis tagastab listi domeen objektidest SourceFile, mis kajastavad üht lähtekoodi faili. Objekti sisuks on lähtekoodi faili nimi ja lähtekoodi faili AST mudelile viite. Iga lähtekoodi faili parsimiseks on kasutatud JavaParseri meetod StaticJavaParser.parse, mis võtab sisendparameetrina Java lähtekoodi faili, mis on esitatud java InputStream objektina. JavaParser klass CompilationUnit kajastab Java lähtekoodi AST mudeli juurtippu. Lisa 3 näitab Java lähtekoodi parsimise koodi. [22]

Samas klassis kus on implementeeritud Java lähtekoodi parsimine asub funktsionaalsus, mis on seotud Java lähtekoodi AST mudeli analüüsiga. See on vajalik peatükis 4.4 käsitletud funktsionaalsuse tagamise jaoks. JavaParser AST mudel koosneb palju erineva tüübi tipudest. Tipu tüübide näited on objekti loomine, klassi väli kasutamine ja klassi meetodi kutsumine. AST mudeli analüüsi jaoks JavaParser teek sisaldab külastaja (*visitor*) klasse. Antud töö kontekstis külastajad on kasutatud konkreetse tüübi tippude

korjamiseks AST mudelist. Joonis 4 näitab koodi kõikide objekti loomise tüübiga tippude korjamiseks.

```
public List<ObjectCreationExpr>
collectObjectCreationExpressions(CompilationUnit compilationUnit) {
    VoidVisitor<List<ObjectCreationExpr>> voidVisitor = new
ObjectCreationExpressionVisitor();
    List<ObjectCreationExpr> objectCreationExpressions = new ArrayList<>();

    voidVisitor.visit(compilationUnit, objectCreationExpressions);

    return objectCreationExpressions;
}

private static class ObjectCreationExpressionVisitor extends
VoidVisitorAdapter<List<ObjectCreationExpr>> {
    @Override
    public void visit(ObjectCreationExpr oce, List<ObjectCreationExpr>
collector) {
        super.visit(oce, collector);
        collector.add(oce);
    }
}
```

Joonis 4. Kood kõikide objekti loomise tüübiga tippude korjamiseks

4.4 Ebaühilduvate Spring API osade kasutamise otsing skannitavast teegist

Esiteks tuleb detailsemalt defineerida mis tüübi ebaühilduvusi otsitakse. Eksisteerib kolm tübi ühilduvust mis eksisteerivad Java programmides [53]:

- lähteühilduvus (source compatibility).
- binaarne ühilduvus (binary compatibility).
- käitumuslik ühilduvus (behavioral compatibility).

Binaarne ühilduvus on detailselt dokumenteeritud Java platformi haldajate poolt ja selle lühidefinitsioon on vigadeta linkimise säilitamise võimalus [53]. Teegi kontekstis binaarne ühilduvus tähendab seda olukorda, kui suvaline Java kood mis on kompileeritud ühe teegi versiooniga ja saab töötada ilma vigadeta selles keskkonnas, kus on paigaldatud teine teegi versioon [54]. Selle olukorra näidiseks on Jakarta servlet API teegi kasutamine Spring rakenduse arendusel, mida paigaldatakse Tomcat serverile. Arenduskeskkonnas see on kättesaadav ainult kompileerimise ajal ja ei ole pakitud koos paigaldatava Spring rakendusega Tomcat serverile. Antud juhul Jakarta servlet API teek on paigaldatud

Tomcat serveril [55] ja sellepärast on võimalik olukord, et Spring rakendus oli kompileeritud ühe Jakarta servlet API teegi versioni vastu ja Tomcat serveril on paigaldatud teine Jakarta servlet API versioon.

Teegid mille jaoks arendatakse rakendust on alati pakitud koos paigaldatava Spring rakendusega. Iga paigaldamisega Spring rakendust taaskord kompileeritakse. See tähendab seda, et teegi versioon mille vastu kompileeritakse rakenduse koodi ja teegi versioon, mis on paigaldatud selles keskkonnas kus rakendus on käivitatud on samad. Sellepärast sisendteekide ühilduvuse kontrollimisel ei ole vaja binaarset ühilduvust kontrollida.

Lähteühilduvus tähendab seda, et kood mis kasutab teegi on kompeleeritav [53]. Juhul kui teegi uuendamisel see kood, mis kasutab teegi enam ei kompeleeri, teegi uus versioon ei ole lähteühilduv vana teegi versiooniga. Seda tüübi ebaühilduvus esineb teekides, milleseid on plaanitud skaneerida arendatava rakendusega. Sellepärast arendatav rakendus peab olema võimeline leida Spring API versiooni 5 osade kasutamist, mis ei ole lähteühilduvad Spring API versiooniga 6.

Käitumuslik ühilduvus tähendab seda, et programm töödeldab sisendit samasuguselt, nagu teine programmi versioon [53]. Seda ühilduvust kontrollitakse testidega. Sellepärast seda ühilduvust ei kontrollitata antud peatükis käsitletud funktsionaalsusest.

Lähtudes antud analüüsist on vaja implementeerida ainult seda funktsionaalsust, mis kontrollib kas teek on lähteühilduv Spring versiooniga 6 või mitte.

Lisa 4 sisaldab koodi, mis implementeerib sisendteegi skaneerimist. Kood võtab sisendina teed mis viitab jar failile, mis sisaldab teegi Java lähtekoodi failid. Programm kasutab peatükkides 4.2 ja 4.3 funktsionaalsust. Prograam kasutab hulk OutdatedApiUsageScanner liidest implementeerivaid objekte. Klassid mis implementeerivad seda liidest implementeerivad mõni konkreetse ebaühilduvuse tüübi otsingu. Näiteks on olemas klass OutdatedConstructorsUsageScanner mis implementeerib mainitud liidest. See klass otsib iga konstruktori kasutamist sisendteegi lähtekoodist ja kontrollib, kas kasutatud konstruktor on võetud Spring API versioonist 5 ja kas see on lähteühilduv Spring API versiooniga 6. Meetod scanLibrary tagastab loendi outdatedApiUsage klassi domeen objektidest. Objekt outdatedApiUsage sisaldab kogu vajalikku informatsiooni leitud ebaühilduva Spring API osa kasutamise kohta. Objekti

outdatedApiUsage sees on lähtekoodi faili nimi, rida ja veerg kus ebaühilduva koodi kasutamine oli leitud, ebaühilduva API osa signatuur ja ebaühilduva API osa muudatuse staatus, mis võib olla kas eemaldatud või muudetud.

4.5 Visuaalse aruanne genereerimine

Selle ülesanne lahendamiseks olid kasutatud Thymeleaf mallide generaator teek ja Bootstrap stiliseeritud komponentide teek. Ülesanne lahendamiseks oli vaja konfigureerida Thymeleaf teegi. On seadistatud mallide asukoht ja malli režiim, mis on seadistatud HTML režiimiks kuna malli mootor on kasutatud HTML lehtede genereerimiseks. On vaja valida ka mallide lahendajat (*template resolver*), mis määrab seda kust mallid HTML lehtede genereerimiseks otsitakse. Mallide lahendajat eksisteerib mitu tüübi. Kõige sobilikum mallide lahendaja tüüp arendatava rakenduse jaoks on ClassLoaderTemplateResolver. See otsib malle klassilaaduri ressursidest ehk resources kataloogist. Teised mallide lahendajad otsivad malle failisüsteemist, URLidest, otseselt antud sisendina tekstist ja veebirakenduse ressursidest (nt. Servlet Context). ClassLoaderTemplateResolver nendest kõikidest on kõige mugavam valik, sest see vajab kõige vähem seadistamist antud rakenduse kontekstis. [56]

Joonis 5. näitab visuaalse aruanne genereerimise koodi. See kood kasutab peatükkis 4.4 käsitletud sisendteegist Spring API vananenud osade kasutamise otsingu. Meetod generateReport võitab sisendparameetrina absoluutset teed, kus visuaalne aruanne on loodud failisüsteemis pärast sisendteegi skaneerimist. Visualse aruanne näidist näitab lisa 5.

```
public void generateReport(String librarySourcesLocation, String
reportLocation) {
    List<OutdatedApiUsage> outdatedApiUsages =
libraryScanner.scanLibrary(librarySourcesLocation);

    Context context = new Context();
    addBootstrapToContext(context);
    context.setVariable("outdatedApiUsages", outdatedApiUsages);
    context.setVariable("libraryName",
extractLibraryNameFromPath(librarySourcesLocation));

    String reportHtml =
ThymeleafConfiguration.templateEngine.process("report", context);

    FileUtils.createTextFile(reportLocation, reportHtml);
}
```

Joonis 5. Visuaalse aruanne genereerimise kood.

4.6 CLI liides

Rakenduse kasutamiseks oli tehtud lihtne CLI kasutajaliides kasutades picocli teegi. Teegi kasutamiseks on vaja paigaldada Java versiooni 21. Ehitatud rakendus on kasutatud käsuga java -jar Tapibara.jar. Tapibara on rakenudse koodnimi. Rakendusel on olemas 2 käsu – dokumentatsioon ja teegi skaneerimine. Dokumentatsiooni käsk on genereeritud picocli teegiga. See käsk näitab juhist, kuidas kasutada teegi. Käsu kasutamiseks on vaja käivitada rakendust parameetriga -h. Skaneerimise käsk võimaldab teostada teegi skaneerimist mille tulemuseks on genereeritud visuaalne aruanne. See käsk vajab kolm parameetri:

- Skaneeritava teegi lähtekoodi sisaldava jar faili asukoht. Parameetri kood on -s või --sources.
- Skaneeritava teegi kompileeritud koodi sisladava jar faili asukoht. Parameetri kood on -b või --binaries.
- Visuaalse aruanne asukoht koos aruanne faili nimega (nt. C:/reports/generatedReport.html). Parameetri kood on -r või --reportLocation.

Lisa 6 sisaldab CLI liidese koodi. CLI klass on programmi sisenemispunkt ja sisaldab main meetodit. Sellepärast samas klassis toimub teegi sõltuvuste seadistamine. Kui kõik kasutatud sõltuvused on seadistatud, alustab teegi skaneerimine ja visualse aruanne genereerimine.

4.7 Testimine

Rakenduse testimiseks oli kirjutatud väike näidisteek, mis kasutab Spring versiooni 5 osi, mis on ebaühilduvad Spring versiooniga 6. Test teegi vastu oli koostatud visuaalne aruanne tehtud rakendusega. Pärast seda töö autor kontollis, et aruanne sisus on olemas kõik kasutatud ebaühilduvad Spring API osad uuemate Spring API versioonidega. Teegi skaneerimise kiiruse testimiseks oli teostatud suuremate avatud lähtekoodiga teekide skaneerimine. Kiiruse mõõtmiseks rakendus oli modifitseeritud, et programmatiliselt arvutada teegi töö kiirust. Programm salvestab aega programmi alguses ja lõpus ning pärast programmi lõpust arvutab vahet programmi alguse ja lõpu vahel. Teek oli testitud

apache-commons-io, apache-commons-text ja apache-commons-lang3 teekide vastu. Testimise järeldused on järgmised:

- apache-commons-io teegi skaneerimine võttis 2.886 sekundit
- apache-commons-text teegi skaneerimine võttis 2.545 sekundit
- apache-commons-lang3 teegi skaneerimine võttis 3.656 sekundit

Testimise tulemused näitavad, et rakendus rahuldab peatükis 3 defineeritud funktsionaalseid ja mittefunktsionaalne nõue rakenduse töö kiiruse kohta, sest visuaalsed aruannet on genereeritud vähem kui 1 minuti jooksul.

4.8 Lõpptulemus

Kasutades peatükis 3 valitud tehnoloogiaid oli arendatud ja testitud uus rakendus. Rakendus on jagatud mitmeteks osadeks vastutusalade järgi. Rakenduse struktuur on selge ja loogiline. Rakendus oskab otsida sisendteegist Spring versiooni 5 API osad, mis ei ole ühilduvad Spring versiooniga 6. Leitud andmete alusel teek oskab genereerida visuaalset aruannet HTML formaadis. Rakendus on kasutatav CLI liidese kaudu. Rakendus on testitud ja testimise tulemused valideerisid, et rakendus rahuldab kehtestatud funktsionaalseid ja rakenduse kiiruse mittefunktsionaalseid nõudeid. Teised rakenduse mittefunktsionaalsed nõuded on ka rahuldatud. Rakenduse kasutamise jaoks on koostatud juhend, rakenduse kasutajaliides on selge ja arusaadav. Rakenduse kood on loogiliselt struktureeritud ja on disainitud niimoodi, et seda oleks võimalik mugavalt testida.

5 Kokkuvõte

Antud töös on detailselt kirjeldatud probleeme, mis on seotud Spring raamistiku tehnoloogiate uuendamise tööde planeerimisega vanade Spring rakenduste jaoks. Probleemi on analüüsitud ja selle lahendamiseks otsustati arendada uus rakendus. Rakenduse jaoks on defineeritud funktsionaalsed ja mittefunktsionaalsed nõuded. Defineeritud nõuete alusel on teostatud tehnoloogiate valik ja plaanitava rakenduse disaini prototüüp. Õigete tehnoloogiate valiku jaoks on teostatud mitmete tehnoloogiate võrdlusanalüüs.

Valitud tehnoloogiate alusel on arendatud uus rakendus CLI kasutajaliidesega. Rakendus on testitud ning testimine valideeris, et rakendus rahuldab funktsionaalseid ja mittefunktsionaalseid nõudeid. Rakendus oskab skaneerida sisendteeki ja otsida selles Spring versiooniga 6 mitteühilduvate Spring API osade kasutamist. Rakendus koostab visuaalse aruande HTML formaadis saadud andmete alusel. Aruannet on võimalik kasutada selleks, et anda mahuhinnangut skaneeritud teegi uuendamisele Spring versioonini 6.

6 Kasutatud kirjandus

[1] VMWare Tanzu, „Upgrading to Spring Framework 5.x,“ 31 08 2021. [Võrgumaterjal]. Available: https://github.com/spring-projects/spring-framework/wiki/Upgrading-to-Spring-Framework-5.x. [Kasutatud 23 11 2023].

[2] VMWare Tanzu, „Upgrading to Spring Framework 6.x,“ 20 11 2023. [Võrgumaterjal]. Available: https://github.com/spring-projects/spring-framework/wiki/Upgrading-to-Spring-Framework-6.x. [Kasutatud 23 11 2023].

[3] J. Delange, „Why Maintaining Software Dependencies is Important and How to Do it Effectively,“ 07 12 2022. [Võrgumaterjal]. Available: https://www.codiga.io/blog/maintain-software-dependencies/. . [Kasutatud 23 11 2023].

[4] VMWare Tanzu, „Spring boot migrator,“ 21 11 2023. [Võrgumaterjal]. Available: https://github.com/spring-projects-experimental/spring-boot-migrator. [Kasutatud 07 12 2023].

[5] enkonix, „What are Functional and Non-Functional Requirements and How to Document These,“ 2023. [Võrgumaterjal]. Available: https://enkonix.com/blog/functional-requirements-vs-non-functional/. [Kasutatud 23 11 2023].

[6] M. Daguerre, „8 Tips for Choosing the Right Software Development Library,“ 19 09 2023. [Võrgumaterjal]. Available: https://www.lagarsoft.com/blog/8-tips-for-choosing-the-right-library. [Kasutatud 27 11 2023].

[7] L. Kühne, „Clirr,“ 2005. [Võrgumaterjal]. Available: https://clirr.sourceforge.net/. [Kasutatud 24 11 2023].

[8] M. Doar, „JDiff - HTML report of API differences,“ 16 04 2013. [Võrgumaterjal]. Available: https://sourceforge.net/projects/javadiff/. [Kasutatud 24 11 2023].

[9] A. Ponomarenko, „Java API Compliance Checker,“ 13 09 2021. [Võrgumaterjal]. Available: https://lvc.github.io/japi-compliance-checker/. [Kasutatud 24 11 2023].

[10] M. Mois, „japicmp github,“ 03 11 2023. [Võrgumaterjal]. Available: https://github.com/siom79/japicmp. [Kasutatud 29 12 2023].

[11] M. Mois, „japicmp,“ 03 11 2023. [Võrgumaterjal]. Available: https://siom79.github.io/japicmp/. [Kasutatud 24 11 2023].

[12] L. Krejci, „Revapi github,“ 08 12 2022. [Võrgumaterjal]. Available: https://github.com/revapi/revapi. [Kasutatud 29 12 2023].

[13] L. Krejci, „Revapi,“ 08 12 2022. [Võrgumaterjal]. Available: https://revapi.org/revapi-site/main/index.html. [Kasutatud 24 11 2023].

[14] D. Smith, „The art of long-term support and what LTS means for the Java ecosystem,“ 09 09 2021. [Võrgumaterjal]. Available: https://blogs.oracle.com/javamagazine/post/java-long-term-support-lts. [Kasutatud 05 12 2023].

[15] Oracle, „Oracle Java SE Support Roadmap,“ 18 09 2023. [Võrgumaterjal]. Available: https://www.oracle.com/java/technologies/java-se-support-roadmap.html. [Kasutatud 05 12 2023].

[16] Bazel, „Dependency Management,“ 23 03 2023. [Võrgumaterjal]. Available: https://bazel.build/basics/dependencies. [Kasutatud 05 12 2023].

[17] JetBrains, „The State of Developer Ecosystem 2023,“ 2023. [Võrgumaterjal]. Available: https://www.jetbrains.com/lp/devecosystem-2023/java/. [Kasutatud 05 12 2023].

[18] Gradle, „Gradle vs Maven Comparison,“ 2023. [Võrgumaterjal]. Available: https://gradle.org/maven-vs-gradle/. [Kasutatud 12 05 2023].

[19] F. Yaremenko, „Maven vs Gradle: How to Choose the Right Build Tool,“ 27 03 2022. [Võrgumaterjal]. Available: https://hackernoon.com/maven-vs-gradle-how-to-choose-the-right-build-tool. [Kasutatud 05 12 2023].

[20] Twilio, „ASTs - What are they and how to use them,“ 11 06 2020. [Võrgumaterjal]. Available: https://www.twilio.com/blog/abstract-syntax-trees. [Kasutatud 27 11 2023].

[21] JavaParser, „JavaParser github,“ 27 11 2023. [Võrgumaterjal]. Available: https://github.com/javaparser/javaparser. [Kasutatud 27 11 2023].

[22] D. v. B. F. T. Nicholas Smith, „JavaParser: Visited,“ Leanpub, 2023.

[23] INRIA, „Spoon github,“ 23 12 2023. [Võrgumaterjal]. Available: https://github.com/INRIA/spoon. [Kasutatud 29 12 2023].

[24] M. M. N. P. C. N. L. S. Renaud Pawlak, „Spoon: A Library for Implementing Analyses and Transformations of Java Source Code,“ *Software: Practice and Experience,* kd. 46, pp. 1155-1179, 2015.

[25] T. Parr, „antrl4 github,“ 15 12 2023. [Võrgumaterjal]. Available: https://github.com/antlr/antlr4. [Kasutatud 29 12 2023].

[26] T. Parr, „ANTLR,“ 07 11 2023. [Võrgumaterjal]. Available: https://www.antlr.org/. [Kasutatud 27 11 2023].

[27] Eclipse Foundation, „Eclipse JDT Core github,“ 25 12 2023. [Võrgumaterjal]. Available: https://github.com/eclipse-jdt/eclipse.jdt.core. [Kasutatud 29 12 2023].

[28] Eclipse Foundation, „JDT Core Component,“ 2023. [Võrgumaterjal]. Available: https://eclipse.dev/jdt/core/. [Kasutatud 28 11 2023].

[29] Thymeleaf, „Thymeleaf github,“ 28 10 2023. [Võrgumaterjal]. Available: https://github.com/thymeleaf/thymeleaf. [Kasutatud 29 12 2023].

[30] Thymeleaf, „Thymeleaf,“ 2023. [Võrgumaterjal]. Available: https://www.thymeleaf.org/index.html. [Kasutatud 28 11 2023].

[31] The Apache Software Foundation, „FreeMarker github,“ 29 12 2023. [Võrgumaterjal]. Available: https://github.com/apache/freemarker. [Kasutatud 29 12 2023].

[32] The Apache Software Foundation, „What is Apache FreeMarker™?,“ 15 01 2023. [Võrgumaterjal]. Available: https://freemarker.apache.org/. [Kasutatud 28 11 2023].

[33] PebbleTemplates, „Pebble github,“ 06 12 2023. [Võrgumaterjal]. Available: https://github.com/PebbleTemplates/pebble. [Kasutatud 29 12 2023].

[34] PebbleTemplates, „Pebble Templates,“ 2023. [Võrgumaterjal]. Available: https://pebbletemplates.io/. [Kasutatud 28 11 2023].

[35] Bootstrap, „Build fast, responsive sites with Bootstrap,“ 2023. [Võrgumaterjal]. Available: https://getbootstrap.com/. [Kasutatud 04 12 2023].

[36] Baeldung, „Read and Write User Input in Java,“ 29 11 2023. [Võrgumaterjal]. Available: https://www.baeldung.com/java-console-input-output. [Kasutatud 29 11 2023].

[37] R. Popma, „Picocli github,“ 20 12 2023. [Võrgumaterjal]. Available: https://github.com/remkop/picocli. [Kasutatud 29 12 2023].

[38] R. Popma, „picocli - a mighty tiny command line interface,“ 27 08 2023. [Võrgumaterjal]. Available: https://picocli.info/. [Kasutatud 29 11 2023].

[39] C. Beust, „JCommander github,“ 24 12 2023. [Võrgumaterjal]. Available: https://github.com/cbeust/jcommander. [Kasutatud 29 12 2023].

[40] C. Beust, „JCommander,“ 21 10 2023. [Võrgumaterjal]. Available: https://jcommander.org/. [Kasutatud 29 11 2023].

[41] VMWare Taznu, „Spring Shell github,“ 24 12 2023. [Võrgumaterjal]. Available: https://github.com/spring-projects/spring-shell. [Kasutatud 29 12 2023].

[42] VMWare Taznu, „Spring Shell,“ 2023. [Võrgumaterjal]. Available: https://spring.io/projects/spring-shell#overview. [Kasutatud 29 11 2023].

[43] The Apache Software Foundation, „Apache Commons CLI github,“ 29 12 2023. [Võrgumaterjal]. Available: https://github.com/apache/commons-cli. [Kasutatud 29 12 2023].

[44] The Apache Software Foundation, „Apache Commons CLI,“ 27 10 2023. [Võrgumaterjal]. Available: https://commons.apache.org/proper/commons-cli/. [Kasutatud 29 11 2023].

[45] R. C. Martin, Clean Code: A Handbook of Agile Software Craftsmanship, Pearson Education, 2008.

[46] Gradle, „Gradle Wrapper Reference,“ 2023. [Võrgumaterjal]. Available: https://docs.gradle.org/current/userguide/gradle_wrapper.html. [Kasutatud 12 05 2023].

[47] Gradle, „Writing Build Scripts,“ 2023. [Võrgumaterjal]. Available: https://docs.gradle.org/current/userguide/writing_build_scripts.html. [Kasutatud 05 12 2023].

[48] git, „gitignore - Specifies intentionally untracked files to ignore,“ 20 11 2023. [Võrgumaterjal]. Available: https://git-scm.com/docs/gitignore. [Kasutatud 05 12 2023].

[49] Gradle, „Settings,“ 2023. [Võrgumaterjal]. Available: https://docs.gradle.org/current/dsl/org.gradle.api.initialization.Settings.html. [Kasutatud 05 12 2023].

[50] Oracle, „What Is a Package?,“ 2022. [Võrgumaterjal]. Available: https://docs.oracle.com/javase/tutorial/java/concepts/package.html. [Kasutatud 12 05 2023].

[51] M. Mois, „Japicmp maven repository,“ 03 11 2023. [Võrgumaterjal]. Available: https://repo1.maven.org/maven2/com/github/siom79/japicmp/japicmp/0.18.3/. [Kasutatud 06 12 2023].

[52] M. Mois, „CLI-Tool,“ 03 11 2023. [Võrgumaterjal]. Available: https://siom79.github.io/japicmp/CliTool.html. [Kasutatud 06 12 2023].

[53] J. Darcy, „Kinds of Compatibility,“ 10 12 2021. [Võrgumaterjal]. Available: https://wiki.openjdk.org/display/csr/Kinds+of+Compatibility. [Kasutatud 06 12 2023].

[54] J. Wang, „BINARY COMPATIBILITY FOR LIBRARY AUTHORS,“ 31 03 2023. [Võrgumaterjal]. Available: https://docs.scala-lang.org/overviews/core/binary-compatibility-for-library-authors.html. [Kasutatud 06 12 2023].

[55] The Apache Software Foundation, „Apache Tomcat Versions,“ 2023. [Võrgumaterjal]. Available: https://tomcat.apache.org/whichversion.html. [Kasutatud 06 12 2023].

[56] Thymeleaf, „Tutorial: Using Thymeleaf,“ 30 07 2023. [Võrgumaterjal]. Available: https://www.thymeleaf.org/doc/tutorials/3.1/usingthymeleaf.html#chaining-template-resolvers. [Kasutatud 06 12 2023].

Lisa 1 – Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks[1]

Mina, Gleb Engalychev

1. Annan Tallinna Tehnikaülikoolile tasuta loa (lihtlitsentsi) enda loodud teose "Sõltuvuste ühilduvuse analüüsitööriista arendus Spring raamistiku versiooniuuenduste jaoks", mille juhendaja on Jaanus Pöial
   1.1. reprodutseerimiseks lõputöö säilitamise ja elektroonse avaldamise eesmärgil, sh Tallinna Tehnikaülikooli raamatukogu digikogusse lisamise eesmärgil kuni autoriõiguse kehtivuse tähtaja lõppemiseni;
   1.2. üldsusele kättesaadavaks tegemiseks Tallinna Tehnikaülikooli veebikeskkonna kaudu, sealhulgas Tallinna Tehnikaülikooli raamatukogu digikogu kaudu kuni autoriõiguse kehtivuse tähtaja lõppemiseni.
2. Olen teadlik, et käesoleva lihtlitsentsi punktis 1 nimetatud õigused jäävad alles ka autorile.
3. Kinnitan, et lihtlitsentsi andmisega ei rikuta teiste isikute intellektuaalomandi ega isikuandmete kaitse seadusest ning muudest õigusaktidest tulenevaid õigusi.

07.12.2023

1 Lihtlitsents ei kehti juurdepääsupiirangu kehtivuse ajal vastavalt üliõpilase taotlusele lõputööle juurdepääsupiirangu kehtestamiseks, mis on allkirjastatud teaduskonna dekaani poolt, välja arvatud ülikooli õigus lõputööd reprodutseerida üksnes säilitamise eesmärgil. Kui lõputöö on loonud kaks või enam isikut oma ühise loomingulise tegevusega ning lõputöö kaas- või ühisautor(id) ei ole andnud lõputööd kaitsvale üliõpilasele kindlaksmääratud tähtajaks nõusolekut lõputöö reprodutseerimiseks ja avalikustamiseks vastavalt lihtlitsentsi punktidele 1.1. ja 1.2, siis lihtlitsents nimetatud tähtaja jooksul ei kehti.

Lisa 2 - API võrdlemise kood

```
public List<JApiClass> compareApi(Path pathToOldApiJar, Path pathToNewApiJar)
{
    Options options = getDefaultJapiCmpOptions();

    JarArchiveComparatorOptions jarArchiveComparatorOptions =
JarArchiveComparatorOptions.of(options);
    JarArchiveComparator jarArchiveComparator = new
JarArchiveComparator(jarArchiveComparatorOptions);

    JApiCmpArchive spring5Archive = new
JApiCmpArchive(pathToOldApiJar.toFile(), "5.0.0.RELEASE");
    JApiCmpArchive spring6Archive = new
JApiCmpArchive(pathToNewApiJar.toFile(), "6.1.1");

    List<JApiClass> jApiClasses;

    try {
        jApiClasses = jarArchiveComparator.compare(spring5Archive,
spring6Archive);
    } catch (Exception e) {
        String errorMessage = String.format("Error while comparing old api %s
to new api %s", pathToOldApiJar, pathToNewApiJar);
        throw new RuntimeException(errorMessage, e);
    }

    return getModifiedAndRemovedObjectsWithChangeStatus(jApiClasses);
}
```

Joonis 6. API võrdlemise kood

Lisa 3 - Java lähtekoodi parsimise kood

```
public List<SourceFile<CompilationUnit>> parseSources(String
pathToLibraryJar) {
    List<SourceFile<CompilationUnit>> librarySourceFiles = new ArrayList<>();

    JarFile jarFile = FileUtils.openJarFile(pathToLibraryJar);

    Enumeration<? extends JarEntry> jarEntries = jarFile.entries();
    while (jarEntries.hasMoreElements()) {
        JarEntry jarEntry = jarEntries.nextElement();
        if (!jarEntry.isDirectory() && jarEntry.getName().endsWith(".java"))
{
            SourceFile<CompilationUnit> sourceFile = parseSourceFile(jarFile,
jarEntry);
            librarySourceFiles.add(sourceFile);
        }
    }

    return librarySourceFiles;
}

private SourceFile<CompilationUnit> parseSourceFile(JarFile jarFile, JarEntry
jarEntry) {
    InputStream jarEntryStream = FileUtils.getJarEntryStream(jarFile,
jarEntry);

    CompilationUnit compilationUnit = StaticJavaParser.parse(jarEntryStream);

    SourceFile<CompilationUnit> sourceFile = new SourceFile<>();
    sourceFile.setName(jarEntry.getName());
    sourceFile.setContents(compilationUnit);

    return sourceFile;
}
```

Joonis 7. Java lähtekoodi parsimise kood

Lisa 4 – Sisendteegi lähtekoodist Spring API versiooniga 6.1.1 ebaühilduvate API osade otsing

```
public List<OutdatedApiUsage> scanLibrary(String pathToLibrarySourcesJar) {
    List<OutdatedApiUsage> outdatedApiUsages = new ArrayList<>();

    List<SourceFile<CompilationUnit>> parsedLibrarySources =
javaLibraryParser.parseSources(pathToLibrarySourcesJar);
    List<JApiClass> outdatedApiParts = compareSpringApis();

    List<OutdatedApiUsageScanner<?>> outdatedApiUsageScanners =
getOutdatedApiUsageScanners(outdatedApiParts);

    for (SourceFile<CompilationUnit> librarySourceFile :
parsedLibrarySources) {
        for (OutdatedApiUsageScanner<?> outdatedApiUsageScanner :
outdatedApiUsageScanners) {
            outdatedApiUsages = Stream.concat(
                    outdatedApiUsages.stream(),
                    outdatedApiUsageScanner.findUsages(librarySourceFile)
                            .stream()).toList();
        }
    }

    return outdatedApiUsages;
}
```

Joonis 8. Sisendteegi lähtekoodist Spring API versiooniga 6 ebaühilduvate API osade otsing

Lisa 5 – Visuaalse aruanne näidis

Library SpringErrorHandlingLib-1.0-sources.jar report

Change status	Used API	File	Location in file
REMOVED	ResponseEntity(org.springframework.http.HttpStatus)	org/example/DefaultExceptionHandler.java	Line: 37 Column: 16
REMOVED	ResponseEntity(java.lang.Object, org.springframework.http.HttpStatus)	org/example/DefaultExceptionHandler.java	Line: 31 Column: 16
REMOVED	ResponseEntity(java.lang.Object, org.springframework.http.HttpStatus)	org/example/GlobalExceptionHandler.java	Line: 23 Column: 16
REMOVED	DefaultUriTemplateHandler()	org/example/GlobalExceptionHandler.java	Line: 64 Column: 56

Joonis 9. Visuaalse aruanne näidis

Lisa 6 – CLI liidese kood

```
@Command(mixinStandardHelpOptions = true)
public class CLI implements Runnable {
    @Option(names = {"-s", "--sources"}, required = true, description = "path
to scanned library source files")
    private static String libraySources;
    @Option(names = {"-b", "--binaries"}, required = true, description =
"path to scanned library compiled code")
    private static File libraryBinaries;
    @Option(names = {"-r", "--reportLocation"}, required = true, description
= "path to HTML report")
    private static String htmlReportLocation;

    @Override
    public void run() {
        setUpJapicmp();
        setUpJavaParser(libraryBinaries);
        setUpThymeleaf();

        HtmlReportGenerator reportGenerator = new HtmlReportGenerator(new
JavaLibraryScanner());
        reportGenerator.generateReport(libraySources, htmlReportLocation);
    }

    public static void main(String[] args) {
        int exitCode = new CommandLine(new CLI()).execute(args);
        System.exit(exitCode);
    }

}
```

Joonis 10. CLI liidese kood