TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Rihard Soodla 185894IADB

Pidevintegratsiooni protsessi kavandamine sardsüsteemide tarkvaraarendusega tegelevas ettevõttes

bakalaureusetöö

Juhendaja: German Mumma
MSc

Tallinn 2021

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: Rihard Soodla

27.03.2021

Annotatsioon

Käesoleva töö raames uuritakse pidevintegratsiooni (ingl k *Continuous Integration*) protsessi väljatöötamist sardsüsteemide tarkvaraarendusega tegelevas ettevõttes.

Pidevintegratsioon on arendusprotsessis järgitavate tavade kogum, mis käsitleb eneses mitmeid praktikaid, kuid selle põhiliseks ideeks on erinevate arendajate poolt tehtud muudatuste pidev ühildamine keskses koodihoidlas. Pidevintegratsiooni praktikate hulka kuuluvad muuhulgas versioonihaldustarkvara kasutamine, tarkvara ehitusprotsessi automatiseerimine, tarkvara automaattestimine ja koodikvaliteedi tagamise tööriistade automaatne käivitamine. Koodimuudatuste pidev integreerimine ja erinevate toimingute automatiseerimine vähendab arendustsüklite pikkust ning vabastab arendajad vajadusest sooritada rutiinseid ja veaohtlikke protseduure manuaalselt.

Töös analüüsitakse ühe Eesti tootearendusettevõtte tarkvaraarenduse protseduure ning nende kitsaskohti, töötatakse välja uus tarkvaraarenduse protsess, mis hõlmab pidevintegratsiooni praktikaid, analüüsitakse erinevaid pidevintegratsiooni lahendusi ning valitakse neist sobivaim.

Pidevintegratsiooni kasulikud omadused on nüüdseks laialdaselt kinnitust saanud, kuid sardsüsteemide tarkvaraarenduse valdkonnas on selle rakendamine vähelevinud. Seega on antud töös tehtud järeldused ja leitud lahendused kasutatavad ka teiste sardsüsteemide tarkvaraarendusega tegelevate ettevõtete kontekstis.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 36 leheküljel, 7 peatükki, 2 joonist, 13 tabelit.

Abstract

Developing a Continuous Integration Process for an Embedded Software Company

This thesis will explore designing a Continous Integration process for an embedded software company.

Continuous integration is a collection of conventions that comprises several practices, but its main goal is integrating the work of multiple developers continuously in a central source code repository. The practices of Continuous Integration include using version control software, automating the software build process, automated testing and automation of code inspection tools. Automating various procedures and integrating code changes frequently reduce the length of development cycles and relieve developers from the need to manually perform routine and error-prone tasks.

Analysis is performed on the software development practices of an Estonian product development company, identifying the bottlenecks and problem areas. A new software development process that encompasses Continuous Integration practises is proposed, analysis is performed on multiple Continuous Integration solutions and a best-suited solution is picked.

The benefits of Continuous Integration are widely accepted, yet its use is not widespread in the domain of embedded software development. Thus, the conclusions arrived at and the solutions found are applicable to other businesses in the domain of embedded software development.

This thesis is written in Estonian and is 36 pages long, including 7 chapters, 2 figures, and 13 tables.

Lühendite ja mõistete sõnastik

API	Ingl k *Application programming interface* - rakendusliides
AS-IS	Olukorra (äriprotsessi) hetkeseis
BPMN	Ingl k *Business Process Model and Notation* - äriprotsesside modelleerimiskeel
CD	Ingl k *Continuous Deployment* - pidevtarne
CI	Ingl k *Continuous Integration* - pidevintegratsioon
CI konveier	Ingl k *Continuous Integration pipeline* - programmaatiline väljendus tarkvaraprojekti ehitamiseks või testimiseks vajalikest sammudest
Clean build	Ingl k *Clean build* - Tarkvara kompileerimine n-ö puhtast kaustastruktuurist. Eesmärk on garanteerida, et väljalaske (ingl k *release*) tegemisel vastaks projekti kaustastruktuuri seis täielikult sellele, milline see on koodihoidlas.
Deemon	Ingl k *Daemon* - taustal töötav protsess multitegumtöö korral, sooritab määratud toiminguid ettemääratud aegadel või teatud sündmuste toimumisel
HIL	Ingl k *Hardware-in-the-loop* - HIL testimine on sardsüsteemide arendamises kasutatav praktika, mille puhul käivitatakse teste testitava riistvara peal ning väliskeskkonna muutusi ja seotud süsteeme simuleeritakse algoritmide abil.
SaaS	Ingl k *Software as a service* - tarkvara teenusena

TO-BE	Olukorra (äriprotsessi) seis pärast väljapakutud muudatuste ellu viimist

Sisukord

1 Sissejuhatus 10
2 Taust 13
2.1 CI teooria 13
2.2 CI sardsüsteemide tarkvaraarenduse valdkonnas 15
2.3 Turul pakutavad CI lahendused 17
2.4 Ettevõtte tutvustus 20
3 Olemasolev süsteem 21
3.1 Olemasoleva infrastruktuuri kirjeldus 21
3.2 AS-IS protsessi kirjeldus 22
3.3 Olemasoleva süsteemi probleemid 24
3.4 Olemasoleva süsteemi probleemide analüüs 25
4 CI protsessi kaardistamine 28
4.1 Kriteeriumid, millele CI lahendus peab vastama 28
4.2 TO-BE CI protsessi välja töötamine 29
5 Sobiva CI lahenduse analüüs ja valimine 32
5.1 Metoodika 32
5.2 Lahenduste analüüs 35
5.3 Sobiva lahenduse valimine 41
5.4 Lahenduse testimine 43
6 Võimalikud edasiarendused 45
7 Kokkuvõte 46
Kasutatud kirjandus 47
Lisa 1 Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks 51
Lisa 2 CI lahenduse kriteeriumite hindamise küsitlus 52
Lisa 3 CI lahenduste kaalutud hinnangute arvutamine 53
Lisa 4 Dockeri süsteemipildi loomise skript 54
Lisa 5 Azure Pipelines ehituskonveieri skript 55

Jooniste loetelu

Joonis 1. AS-IS protsessi töövoog 23
Joonis 2. TO-BE protsessi töövoog..... 31

Tabelite loetelu

Tabel 1. MoSCoW meetodi tulemuste teisendamine numbrilisele skaalale 33
Tabel 2. Kriteerium K1 hindamisskaala 33
Tabel 3. Kriteerium K2 hindamisskaala 33
Tabel 4. Kriteerium K3 hindamisskaala 34
Tabel 5. Kriteerium K4 hindamisskaala 34
Tabel 6. Kriteerium K5 hindamisskaala 35
Tabel 7. Hinnangud Jenkins CI lahendusele 36
Tabel 8. Hinnangud TeamCity CI lahendusele 37
Tabel 9. Hinnangud BuildBot CI lahendusele 38
Tabel 10. Hinnangud GitLab CI/CD lahendusele 39
Tabel 11. Hinnangud Azure Pipelines CI lahendusele 40
Tabel 12. Küsitluse tulemused 42
Tabel 13. Lahendustele kaalutud hinnangute andmine 42

1 Sissejuhatus

Agiilsed tarkvaraarenduse metoodikad on klassikalist koskmudelil põhinevat lähenemist välja tõrjumas ning on saanud uueks normiks [1]. Põhjuseid selleks ei pea kaugelt otsima – agiilsetel projektidel on suurem tõenäosus edule [2].

Kuigi agiilsed metoodikad on kasutatavad väga erinevates tarkvaraprojektides, ei ole need sardsüsteemide arenduse valdkonnas laia rakendust leidnud. Sellise olukorra üheks põhjuseks on sardsüsteemide arenduse valdkonna fundamentaalsed eripärad, mis raskendavad agiilsete väärtuste, põhimõtete ja praktikate rakendamist. [3] Olenemata valdkonna spetsiifikast tulenevatest probleemidest, ei leidnud Kaisti jt [4] põhjust, miks ei võiks agiilseid metoodikaid rakendada sardsüsteemide arenduse valdkonnas.

Agiilseid metoodikaid on mitmeid, kuid omasuguste seas mahukaima ja vanima uuringu "State of Agile" väitel on kolmeks enimkasutatavaks agiilseks inseneripraktikaks ühiktestimine (ingl k *unit testing*), ettevõtteüleste koodistandardite kasutamine ning CI rakendamine [5]. Käesolevas töös keskendutaksegi eelkõige nende praktikate rakendamise uurimisele ühes Eesti tootearendusettevõttes. Ettevõtte peamiseks tegevusvaldkonnaks on elektroonikatoodete arendusteenuse pakkumine, mis hõlmab endas ka sardsüsteemide tarkvaraarendust. Töös analüüsitakse ettevõtte tarkvaraarenduse protseduure ning nende kitsaskohti, selgitatakse välja ettevõttele sobivaim CI protsess, analüüsitakse erinevaid CI lahendusi ning valitakse neist sobivaim.

Teema valiku põhjustas asjaolu, et uurimisalune ettevõte on moderniseerimas oma infrastruktuuri. Et vältida töö seiskumist, viiakse suuremahulised tööprotsessi muudatused läbi astmeliselt. Ettevõtte infrastruktuuri moderniseerimise protsess koosneb kolmest faasist. Esimeses, praeguseks hetkeks lõpustaadiumisse jõudnud faasis, võeti kasutusele Azure DevOps tarkvaraarendusplatvorm ning juurutati selle alamplatvormi Azure Repos kasutamine kõikide ettevõtte tarkvaraprojektidega seonduvate lähtekoodifailide hoidlana. Uuendati ka ettevõtte standardset ehitussüsteemi, minnes üle Makefile põhiselt süsteemilt CMake'le. Teises faasis on plaanis ettevõttes juurutada CI praktikaid ning võtta kasu-

tusele mõni turul pakutavatest CI läbiviimiseks mõeldud automatiseerimisserveritest (CI lahendustest). Soovitakse senisest rohkem pöörata tähelepanu arendatava tarkvara ühiktestimisele ja erinevate seni manuaalsete protseduuride automatiseerimisele. Kolmandas faasis on plaanis ettevõttes juurutada HIL-testimist.

Vajaduse teises ning kolmandas faasis planeeritavate muudatuste järgi on põhjustanud ettevõttes arendatavate projektide iseloomu muutumine. Kui varasemalt olid projektid loomult reeglina ühekordsed ning lõppesid ettevõtte jaoks toote üleandmisega, siis nüüdseks on ettevõttele tekkinud kogum jooksvalt arendatavaid suuremahulisi projekte. Enne selliste projektide tekkimist pole olnud CI praktikate rakendamine kuluefektiivne. Ühtlasi on ka arenduspraktikad aja jooksul muutunud. Hetkel kui praegu kasutusel olevat protsessi disainiti, polnud agiilsed metoodikad veel suuremat populaarsust kogunud. Kokkuvõtteks – CI praktikate juurutamise tulemusena soovitakse tõsta arendusprotsessi efektiivsust ja pakutava teenuse kvaliteeti ning saavutada seeläbi konkurentsieelis.

Antud töös tegeletakse teises moderniseerimisfaasis planeerivate muudatuste kavandamisega. Kuna kolmandas faasis plaanitavad muudatused sõltuvad otseselt teise faasi juurutamisest, siis ei käsitleta töös HIL-testimist.

CI praktikad on tänaseks päevaks väga hästi dokumenteeritud ning kirjeldatud, kuid sardsüsteemide tarkvaraarendusele spetsiifilist kirjandust leidub vähe. Seega on üheks töö eesmärgiks uurida CI rakendamist sardsüsteemide tarkvaraarenduse kontekstis.

Töö teises peatükis tutvustatakse pidevintegratsiooni kontseptsiooni, selle põhimõtteid ning kasutegureid ning uuritakse CI rakendamist sardsüsteemide tarkvaraarenduse valdkonnas. Antakse ülevaade erinevatest turul pakutavatest CI lahendustest. Peatüki lõpus tutvustatakse uurimisalust ettevõtet.

Kolmandas peatükis tutvustatakse ettevõttes juba kasutusel olevat tarkvaraarendusega seotud infrastruktuuri. Kirjeldatakse ära AS-IS tarkvaraarendusprotsess ning analüüsitakse selle kitsaskohti ja puuduseid ning defineeritakse probleemid, mille uus protsess lahendama peab.

Neljandas peatükis kaardistatakse uurimisalusele ettevõttele sobivaim CI protsess. Defineeritakse kriteeriumid, millele valitav CI lahendus vastama peab. Töötatakse välja TO-BE arendusprotsess, mis hõlmab ka pidevintegratsiooni praktikaid.

Viiendas peatükis valitakse teises peatükis tutvustatud turul pakutavate CI lahenduste seast välja uurimisalusele ettevõttele sobivaim lahendus, eelnevalt tutvustatakse ka valiku tegemiseks kasutatavat metoodikat. Viiakse läbi valitud lahenduse testimine.

Kuuendas peatükis tuuakse välja võimalikud edasiarendused.

2 Taust

Peatükis käsitletakse CI kontseptsiooni, tuuakse välja peamised CI põhimõtted ja praktikad ning uuritakse CI rakendamist sardsüsteemide tarkvaraarenduse valdkonnas. Antakse lühiülevaade valitud turul pakutavatest CI lahendustest. Tutvustatakse ka antud bakalaureusetöös analüüsitavat ettevõttet.

2.1 CI teooria

CI on tarkvaraarenduse praktika, mille järgi tiimi liikmed ühildavad oma tööd sagedasti, tavaliselt integreerib iga tiimi liige koodimuudatused vähemalt kord päevas – seega integreeritakse muudatusi mitu korda päevas. Selleks, et tuvastada integratsioonivead võimalikult varajaselt, verifitseeritakse iga integratsioon automatiseeritud ehitusprotsessi ja testimise abil. Mitmed tarkvaraarendustiimid leiavad, et selline lähenemine vähendab oluliselt erinevaid integratsiooniprobleeme ning võimaldab tiimil arendada kiiremini sidusat tarkvara. [6]

Humble ja Farley [7] sõnul on CI edukaks rakendamiseks vaja kolme asja:

1. Versioonihaldus – kõik projekti komponendid (lähtekood, testid, ehitusskriptid jne) peavad olema versioonihalduse all, ühes koodivaramus.
2. Automatiseeritud ehitusprotsess – projekti ehitamine peab olema võimalik käsurealt.
3. Tiimi nõusolek – CI on praktika, mitte tööriist. See nõuab arendustiimilt pühendumust ja distsipliini. Igaüks peab versioonihaldussüsteemi lisama sagedasi, inkrementaalseid muudatusi, ning nõustuma, et kõrgeima prioriteediga ülesanne on nende muudatuste parandamine, mis rakenduse mingil põhjusel katki teevad.

Kuigi CI rakendamine on võimalik ka ilma selle jaoks mõeldud tööriistade abita, on CI serveri abil CI praktikate automatiseerimine efektiivne ning soovituslik viis probleemile

lähenemiseks [8]. Antud töö kontekstis ongi valitud lähenemine, kus CI praktikate rakendamiseks kasutatakse abivahendina CI automatiseerimisserverit.

Arutledes CI väärtuse üle arendusprotsessi komponendina, toob Duvall [8] välja viis põhilist väärtusaspekti:

1. Riskimaandus – koodimuudatuste sage integreerimine võimaldab vigade varajasemat avastamist ja parandamist. Kuna CI integreerib, käivitab automaatteste ja inspektsioonitööriistu mitmeid kordi päevas, on suurem tõenäosus, et vead avastatakse nende tekkimise hetkel (vahetult peale koodivaramusse muudatuste lisamist). Automatiseeritud integratsiooniprotsessi kasutamine tähendab ka seda, et tarkvaraprojekti "tervislik seisund" on mõõdetav ning igal hetkel teada. Üheks jälgitavaks mõõdikuks võib olla näiteks koodi keerukus. Tarkvara korduv ehitamine ja testimine selleks mõeldud CI serveris, *clean build* põhimõttel, vähendab tehtavaid eeldusi ning kindlustab ehitusprotsessi sõltumatuse arenduskeskkonnast.
2. Rutiinsete tegevuste vähendamine – rutiinsete tegevuste vähendamine säästab aega, kulusid ja jõupingutusi. Rutiinseid tegevusi võib leiduda kõikides projektiga seotud toimingutes: koodi kompileerimisel, testimisel, inspekteerimisel, paigaldamisel ja tagasiside andmisel. CI automatiseerimine lubab arendajatel keskenduda kõrgema väärtusega töö tegemisele.
3. Tarnitava tarkvara loomine – CI võimaldab misiganes ajahetkel väljastada tarnitavat tarkvara. Kui värske koodimuudatuse tagajärjel tekib mõni viga, parandatakse see niipea kui võimalik. See tähendab, et projekt veedab vähem aega n-ö katkises seisus ning väljalasked saavad toimuda ilma eelneva integreerimisfaasita.
4. Parem ülevaade projektide olukorrast – CI võimaldab märgata trende ning teha efektiivseid otsused. CI lahenduste abil on võimalik kiiresti hinnata projekti hetkeseisu ja kvaliteedimõõdikuid, eemaldades vajaduse teha selliseid hinnanguid manuaalselt ilma toetavate andmeteta.

5. Enesekindluse toomine arendusprotsessi – tarkvaraprojekti tiim teab, et iga automatiseeritud ehitusprotsessi käigus kontrollitakse koodi korrektsust automaattestide abil ning verifitseeritakse koodi vastavust projekti disaini- ja koodistandarditele. Kuna CI süsteem teavitab arendajad kohe, kui miski valesti läheb, saavad tiimiliikmed muudatusi teha suurema enesekindlusega.

2.2 CI sardsüsteemide tarkvaraarenduse valdkonnas

Lähtuvalt asjaolust, et CI näol on tegemist ühe põhilisima agiilse inseneripraktikaga ning sellest, et agiilsed metoodikad pole sardsüsteemide arenduse valdkonnas laialt kasutusel, võib väita, et CI on küll serveritarkvara maailmas üsna laialt levinud, kuid sardsüsteemide tarkvara valdkonnas kasutatakse seda suhteliselt vähe [3], [5].

Lwakatare jt sõnul on serveritarkvara arendamine sardsüsteemide tarkvara arendamisest fundamentaalselt erinev – sardsüsteemide tarkvara arendamise valdkonnas on tarkvara vaid üks komponent riistvara kõrval. [9]

Artiklis *"Towards DevOps in the Embedded Systems Domain: Why is It so Hard?"* on Lwakatare jt välja toonud neli põhilist raskuspunkti *DevOps* rakendamisel sardsüsteemide tarkvaraarenduse valdkonnas [9]:

- sõltuvus riistvarast ja vajadus ühilduda mitmete erinevate tarkvaraversioonidega;
- piiratud arusaam keskkondadest, milles kliendid tooteid kasutavad;
- puudulik tehnoloogia uute funktsionaalsuste automaatseks ja töökindlaks tarnimiseks;
- puudulik andmete kogumine juba kasutusel olevatest toodetest.

Välja toodud raskuspunktidest on CI praktikate rakendamise juures ilmselt olulisim sardsüsteemide tarkvara sõltuvus riistvarast. Selle sõltuvuse tõttu pole võimalik teostada integratsioonitestimist ilma riistvarata. Riistvara ja tarkvara testimist komplektina nimetatakse HIL-testimiseks. HIL-testimine on olemuselt suhteliselt keeruline ning vajab edu-

kaks läbiviimiseks hästi läbimõeldud seadistust. Mõistagi on antud seadistus iga projekti juures suhteliselt erinev, sõltudes vastava projekti funktsionaalsetest nõuetest, võimalikest liidestest ning kasutatava riistvara eripäradest. Üheks HIL-testimise juurutamise eelduseks on olemasolev infrastruktuur n-ö tavapärase CI läbiviimiseks.

Ilma HIL-testimiseta pole võimalik sardsüsteemide tarkvara integratsioonitestimine, kuid on võimalik testida tarkvara funktsionaalseid nõudeid, jättes testimata osad, mis sõltuvad otseselt riistvarast. Sedasorti praktikat tuntakse eelkõige ühiktestimise nime all. Vastavalt sissejuhatuses paika pandud töö vaatlusalale ning ettevõtte infrastruktuuri moderniseerimisprotsessi faasidele ei keskenduta antud töös sardsüsteemide tarkvara integratsioonitestimisele (HIL-testimisele). Testimisest rääkides mõeldakse eelkõige ühiktestimist.

Üheks oluliseks raskuseks CI rakendamisel sardsüsteemide tarkvaraarenduse valdkonnas on ka erinevate projektide omapärasus. Erinevad projektid on suunatud erinevatele platvormidele, kasutatakse erinevaid programmeerimiskeeli, *build*-süsteeme ja krosskompilaatoreid. Sedasorti keerukus serveritarkvara arendamise maailmas tüüpiliselt puudub, kuna reeglina arendatakse tarkvara üldkasutatavale x86-64 arhitektuurile. Samuti on ka serveritarkvara valdkonna arendusökosüsteemid (nt Java või C# programeerimiskeelte puhul) oluliselt standardsemad ja universaalsemad – see soosib CI kasutamist.

Uurimisaluses ettevõttes kehtib ka põhimõte, et tarkvaraprojekt tuleb täielikult ise lähtekoodist kokku kompileerida – eelkompileeritud binaarkujul olevate teekide ja programmide kasutamine on lubatud vaid siis, kui lähtekood pole saadaval. Reeglina tuleb tarkvara lähtekoodist krosskompileerida, kuna kood tuleb optimeerida vastvalt konkreetsele riistvarale. Sellise optimeerimise eesmärk võib olla näiteks toote aku kestvuse pikendamine või püsivarapaketi suuruse vähendamine. Põhimõtte teiseks põhjuseks on asjaolu, et tihtipeale pole vastavat tarkvara sihtplatvormile eelkompileerituna saadaval. Kolmandaks, isegi kui eelkompileeritud tarkvara on saadaval, võib see mingi hetk muutuda kättesaamatuks, mistõttu tekitab selle kasutamine teatud riski. Neljandaks võimaldab tarkvara täielik lähtekoodist kompileerimine vajaduse korral kergesti välistes teekides ja programmides muudatusi läbi viia. Sedasorti põhimõte on tarkvaraarenduses suhteliselt unikaalne sard-

süsteemide valdkonnale ning toob endaga kaasa lisakeerukust ja mõjutab oluliselt projektide kompileerimiseks kuluvat aega.

Lisaks tehnilise iseloomuga keerukuskohtadele on ilmselt üheks põhjuseks, miks CI pole sardsüsteemide tarkvaraarenduse valdkonnas laialdaselt levinud ka asjaolu, et sardsüsteemide valdkonna projektid on tihtipeale lühema elutsükliga kui muude tarkvaraarenduse valdkondade projektid. Kuna automatiseerimisprotsesside algne seadistus võib olla keerukas ning aeganõudev, sõltub CI rakendamise kuluefektiivsus suuresti projekti kestvusest.

Kokkuvõtteks võib öelda, et võrreldes serveritarkvara arendamise valdkonnaga on CI rakendamine sardsüsteemide tarkvaraarenduse valdkonnas nüansirohkem ning nõuab nii lahenduse valikul kui ka protsessi väljatöötamisel sügavamat analüüsi ja plaaneerimist.

2.3 Turul pakutavad CI lahendused

Toetudes kahele artiklile, mis käsitlevad CI tööriistade valimist sardsüsteemide tarkvaraarenduse kontekstis, on autor valinud välja neli CI lahendust. Valituks osutusid need lahendused, mis artiklite autorite hinnangul sobivad sardsüsteemide tarkvaraarenduseks. Nendeks on Jenkins, TeamCity, BuildBot ja GitLab CI/CD. [10], [11]

Lisaks eelmainitud lahendustele otsustas autor lisada valikusse lahenduse Azure Pipelines. Selle otsuse põhjuseks oli asjaolu, et uurimisaluse ettevõtte hetkel kasutatavaks versioonihaldustarkvaraks on Azure DevOps Services. Kuna Azure Pipelines on Azure DevOps Services komponent, siis võib eeldada, et nende lahenduste integreerimine on võrreldses kolmandate osapoolte lahendustega lihtne ja vähese ressursikuluga.

Järgnevalt on välja toodud eelmainitud viie turul pakutava CI lahenduse lühikirjeldused. Töö edasistes peatükkides kirjeldatav uurimisalusele ettevõttele sobivaima lahenduse valimine tehakse selles peatükis tutvustatavate lahenduste seast.

Jenkins

Jenkins on iseseisev vabavaraline (MIT litsents) automatiseerimisserver, mille abil on võimalik automatiseerida igat sorti ülesandeid, mis seonduvad tarkvara ehitamise, testimise, üleandmise või paigaldamisega. Jenkins on tasuta kasutatav tarkvara, mida on võimalik paigaldada nii Linuxile, Windowsile kui ka macOSile. Jenkinsi põhiliseks tunnusjooneks on Jenkins Pipeline, mis kujutab endast kogumit pistikprogramme, mis võimaldavad Jenkinsisse integreerida CI konveiereid. Konveierid defineeritakse Jenkinsis Groovy programmeerimiskeelel põhineva Pipeline valdkonnaspetsiifilise keele abil. Jenkins Pipeline'i definitsioon kirjutatakse tekstifaili (*Jenkinsfile*), mis tüüpiliselt lisatakse ka versioonihaldusesse. [12]–[14]

TeamCity

JetBrains TeamCity on minimaalsete seadistusnõuetega võimas ja kasutajasõbralik CI/CI server [15]. TeamCity on toetatud nii Linuxil, Windowsil kui ka macOSil [16]. Teamcity on saadaval nii tasuta kui ka tasulise versioonina. Tasuta versioonis on kogu toetatud funktsionaalsus olemas, kuid piiratud on võimalike ehituskonfiguratsioonide ja -agentide arv. [17]

Buildbot

Buildbot on vabavaraline (GPLv2 litsents) Pythonis kirjutatud CI raamistik. See koosneb peadeemonist (ingl k *master daemon*) ja potentsiaalselt mitmest töölisdeemonist (ingl k *worker daemon*), mida tüüpiliselt jooksutatakse erinevatel masinatel. Peadeemon hõlmab endas veebiserverit, mis võimaldab lõppkasutajal käivitada ehitusprotsesse ja Buildbot instantsi konfigureerida. Peadeemon tegeleb ka ehitusprotsesside tööliste vahel hajutamisega. [18] Buildbot raamistiku konfigureerimine ning ehitusprotsesside kirjeldamine toimub läbi Pythoni skriptide, mistõttu võimaldab see kirjeldada mistahes ehitusprotsesse, sõltumata projektis kasutatavatest programeerimiskeeltest või muudest projekti omadustest [19].

Gitlab CI/CD

Gitlab on täielikult integreeritud tarkvaraarendusplatvorm, mis hõlmab endas muuhulgas versioonihaldussüsteemi, probleemijälgimissüsteemi (ingl k *issue tracker*), sissehitatud funktsionaalsust CI rakendamiseks ja palju muud [20]. Lähteülesandest sõltuvalt keskendutakse Gitlab tarkvara vaatlemisel selle alamkomponendile Gitlab CI/CD, mis on Gitlab tarkvarasse sisseehitatud tööriist. Gitlab CI/CD võimaldab defineerida konveiereid, mille abil on võimalik defineerida tarkvara tarnerprotsessi sammud ning reeglid, mille alusel antud samme täidetakse. Konveieri defineerimine toimib YAML serialiseerimiskeeles kirjutatud tekstifaili abil, mis paigutatakse projekti koodihoidla juurkausta. [21], [22] Enamus Gitlabi funktsionaalsusest on saadaval tasuta versioonis. Tasulised versioonid hõlmavad endas spetsiifilisemaid funktsionaalsusi, mis on suunatud eelkõige ettevõtte mänedžeridele või juhtkonnale. Olemas on kaks Gitlab distributsiooni: Community Edition (CE) ja Enterprise Edition (EE). Gitlab CE on MIT litsentsi all. Gitlab EE kasutab sama tuuma, mis CE versioon (MIT litsents) ning EE-spetsiifiline kood on kommertslitsentsi all, kuid vabalt kättesaadav (ingl k *source-available*). Gitlab on saadaval nii *SaaS* (ingl k *Software as a service*) variandis kui ka isemajutatuna. [23]

Azure Pipelines

Azure Pipelines on Azure DevOps tarkvaraarendusplatvormi komponent, mis võimaldab tarkvaraprojektide automaatset ehitamist ja testimist sõltumata projekti tüübist või kasutatavast programmeerimiskeelest. Privaatsete projektide korral on Azure Pipelines kasutamine tasuline, kuid esimesed 1800 tööminutit on iga kuu tasuta. [24] CI/CD konveierite definieerimiseks kasutatakse Azure Pipelines keskkonnas YAML keeles kirjutatud tekstifaili, mis asub koodihoidla juurkaustas [25]. Azure DevOps on saadaval nii *SaaS* variandis (Azure DevOps Services) kui ka isemajutatuna (Azure DevOps Server), mõlemal puhul on tegemist omandtarkvaraga [26].

2.4 Ettevõtte tutvustus

Uurimisalune ettevõte on multidistsiplinaarne tootearendusettevõte, mille peamiseks tegevusvaldkonnaks on elektroonikatoodete arendus ja tootmine. Ettevõte on oma valdkonnas tegutsenud üle 20 aasta. Ettevõtte eesmärgiks on pakkuda klientidele täisteenust, hõlmates kogu elektroonikatoote arenduse juurde kuuluvat – aidata muuta klientide ideed reaalseteks toodeteks. Lisaks klientidele pakutavale teenusele arendab ettevõte väiksemas mahus ka oma tooteid.

Ettevõtte projektid on olemuselt mitmekesised ning tarkvara arendatakse erinevates programeerimiskeeltes – põhiliselt on nendeks kas C, C++ või Java, vähesemal määral kasutatakse C# ja JavaScript programeerimiskeeli. Tarkvara arendatakse mobiilseadmetele (Android, iOS), personaalarvutitele (Windows, Linux, macOS), mikrokontrolleripõhistele sardsüsteemidele (Cortex-M, AVR, i8051) ja ka aplikatsiooniprotsessoripõhistele sardsüsteemidele (Cortex-A, ARM9, MIPS, x86).

3 Olemasolev süsteem

Peatükis tutvustatakse ettevõttes juba kasutusel olevat tarkvaraarendusega seotud infrastruktuuri. Kirjeldatakse ära AS-IS tarkvaraarendusprotsess ning analüüsitakse selle kitsaskohti ja puuduseid, ning defineeritakse probleemid, mille uus protsess lahendama peab.

3.1 Olemasoleva infrastruktuuri kirjeldus

Järgnevalt kirjeldatakse ettevõtte infrastruktuuri tarkvaraarendusega seotud komponente.

Versioonihaldusplatvorm

Ettevõte on parasjagu oma infrastruktuuri moderniseerimas. Varasemalt oli kasutusel isemajutatud ettevõttesisene Git versioonihaldussüsteemi server. Tänaseks on enamus arendus- või hooldusfaasis olevaid projekte üle viidud Azure Devops tarkvaraarendusplatvormile. Põhiliselt kasutatakse Azure Devops platvormi alamkomponenti Azure Repos (Git koodihoidlad). Azure Devopsi kasutatakse projektide lähtekoodi hoiustamiseks ning koodimuudatuste integreerimiseks läbi tõmbetaotluste (ingl k *pull request*). Ettevõtte tavad näevad ette, et arendajatel pole võimalik koodimuudatusi otse koodihoidla peaharusse (ingl k *master branch*) teha. Iga tehtav muudatus tuleb peaharusse mestida (ingl k *merge*) läbi tõmbetaotluse, millele peab pärast koodi läbivaatuse (ingl k *code review*) sooritamist heakskiidu andma vähemalt üks arendaja peale muudatuste autori.

Failiserver

Ettevõttes on kasutusel keskne failiserver, kus hoiustatakse erinevaid ettevõtte ja disainitööga seonduvaid faile. Tarkvaraarenduse valdkonna seisukohast on olulisimad kaustad *Release* ja *Install*. *Release* kaustas hoitakse kõiki klientidele väljasaadetavaid faile, sinna kuuluvad näiteks binaarfailid, dokumentatsioonifailid ja lähtekoodi väljalaskepakid (ingl k *source release packages*). *Install* kaustas hoitakse muuhulgas arendustööks kasutata-

vaid arvutiprogramme (sealhulgas hulgaliselt erinevaid krosskompilaatoreid) ja erinevate tarkvaraprojektide lähtekoodi väljalaskepakke.

3.2 AS-IS protsessi kirjeldus

Järgnevalt kirjeldatakse tüüpilist tarkvaraarendusprotsessi arendaja vaatest sellisena, nagu see praegu ettevõttes on. Protsessi sisendiks on arendajale antud tööülesanne.

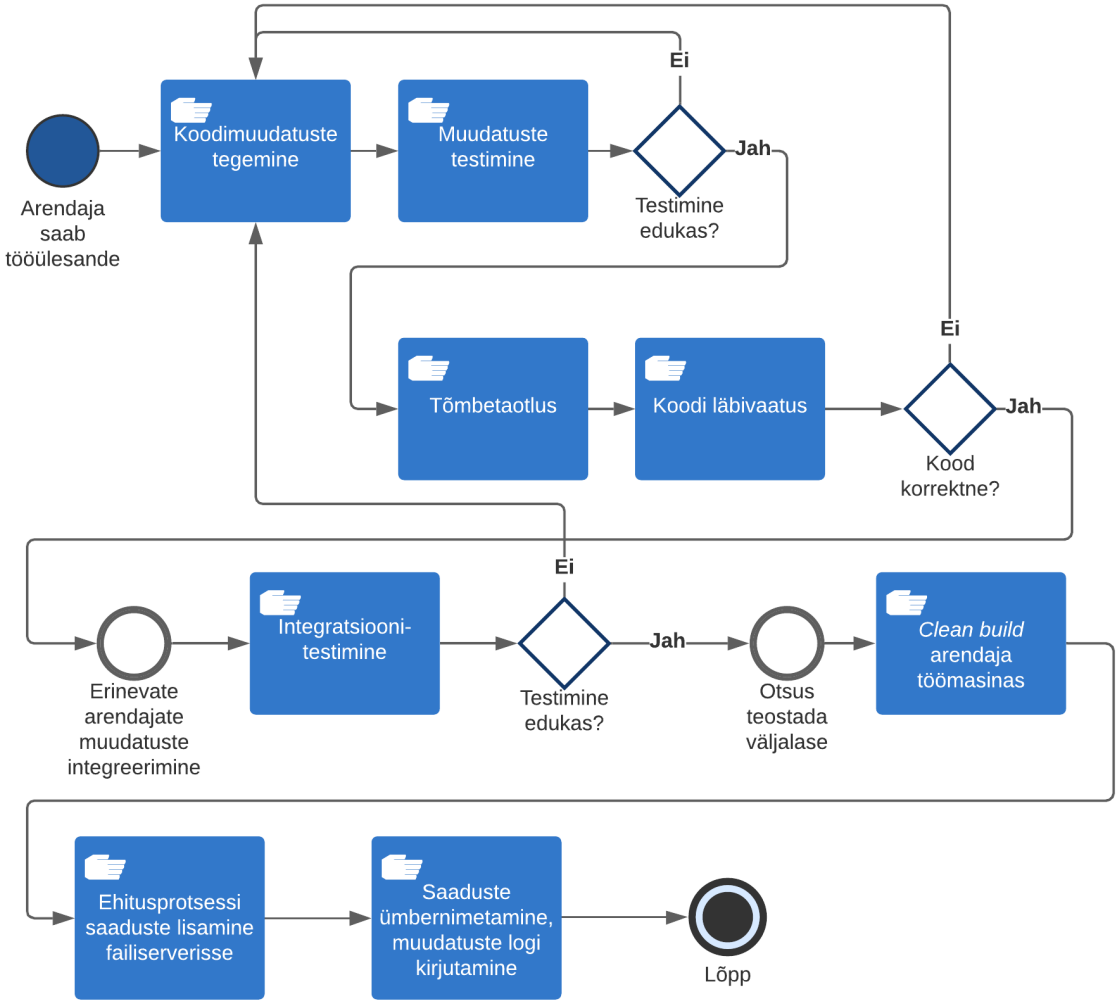
Arendaja kloonib (ingl k *clone*) endale töömasinasse lähtekoodivaramust vastava projekti koodihoidla. Vajadusel laeb arendaja enda töömasinasse projekti kompileerimiseks kasutatava (kross)kompilaatori, mis asub ettevõtte failiserveri *Install* kaustas. Arendaja viib läbi koodimuudatusi, teostades arendustööd n-ö isiklikus koodihoidla harus. Koodianalüüsi- ja vormindamistööriistade kasutamine ning automaattestide (olemasolul) jooksutamine toimub arendaja enda äranägemise järgi. Olles koodimuudatuste tegemise lõpetanud, testib arendaja tarkvara toimimist, teostab koodihoidlasse *commit*'id, laeb muudatused eraldi haruna Azure DevOps koodivaramusse ning teostab peaharusse tõmbetaotluse. Pärast mõnelt teiselt arendajalt tõmbetaotlusele heakskiidu saamist mestitakse muudatused koodihoidla peaharusse. Kui läbivaatust teostav arendaja leiab koodimuudatustest vigu või teeb nende kohta soovitusi, siis viibib taotluse jõustumine kuni vead on parandatud ning soovitustega tegeletud. Tsüklit korratakse, kuni tarkvara on valmis väljalaskeks (ingl k *release*). Enne väljalaske teostamist testitakse tarkvara uuesti, et vältida erinevate muudatuste integreerimisest tekkinud vigade esinemist.

Kuna ettevõtte väljalaskeprotseduur näeb ette väljalasete ehitamist *clean build* põhimõttel, siis kloonib arendaja enda arvutisse projekti koodihoidla vastavast harust, kust väljalaset teha soovitakse. Seejärel käivitab arendaja tarkvara ehitusprotsessi. Tulenevalt sellest, et tegemist on *clean build*'iga, võtab see aega oluliselt kauem, kui juba ehitatud projekti uuesti kompileerimine[1]. Protsessi lõppedes tõstab arendaja ehitusprotsessi saadused, näiteks täitmisfailid (ingl k *executable files*) või püsivarapaketid, ettevõtte failiserverisse vastava projekti kausta. Sõltuvalt projektist ja konkreetsest ehitussüsteemist võib olla

[1] Sel puhul saab ehitussüsteem reeglina aru, millised lähtekoodifailid on muutunud ning kompileerib ainult neid

vajalik ehitusprotsessi saaduste ümbernimetamine (nt versiooninumbri lisamiseks). Reeglina lisab arendaja projekti kausta või vastava väljalaske versioonihalduse sildi (ingl k *tag*) külge ka muudatuste logi (ingl k *changelog*).

Protsessist ülevaate saamiseks on loodud töövoog joonisel 1. Töövoo kujutamiseks on kasutatud BPMN[1] märgistikku. Nagu näha, on iga protsessi käigus sooritatav ülesanne kas manuaalne või manuaalselt käivitatav.



Joonis 1. AS-IS protsessi töövoog.

[1] `https://camunda.com/bpmn/reference`

3.3 Olemasoleva süsteemi probleemid

Olemasolev süsteem toimib, kuid on olemuselt manuaalne ja ebatõhus ning jätab palju ruumi inimvigade tekkimiseks.

Reeglina töötavad arendajad n-ö isiklike versioonihalduse harude peal ning muudatusi integreeritakse suhteliselt harva. Mida harvemini koodimuudatusi integreeritakse, seda tõenäolisem on mitme integratsioonikonflikti esinemise risk [8].

Integreerimise üheks suurimaks pudelikaelaks on muudatustele heakskiidu andva arendaja poolt koodi läbivaatusele kuluv aeg. Hetkel peab läbivaatust teostav arendaja veenduma lisaks koodi funktsionaalsele korrektsusele muuhulgas selles, et muudatuste jõustumisel tarkvara üldse kompileeruks, et kood vastaks ettevõttesisestele koodistiili nõuetele, ning et muudatuste tagajärjel ei tekiks mõnda vähemärgatavat tarkvaraviga. Eriti keeruline on koodi läbivaatuse käigus avastada mäluvigu (ingl k *memory errors*) ja mitmelõimelisusega seotud vigu (ingl k *concurrency bugs*).

Pärast muudatuste integreerimist on vajalik nende testimine, et veenduda tarkvara ootuspärases käitumises. Projektide puhul, millel puuduvad automaattestid (valdav enamus), tähendab see laialdast manuaalset testimist, mis on ajakulukas. Manuaalse testimise puhul on ka tõenäoline, et ei testita süsteemi kõiki aspekte ning vead jäävad märkamata.

Tulenevalt ettevõtte väljalaskeprotseduuri põhimõtetest võib antud protseduur koodimahukamate projektide, nt Linux kernel-i või Androidi kompileerimisel aega võtta tunde. Selle aja vältel võib arendaja tööarvuti olla niivõrd koormatud, et muu ressursinõudliku töö tegemine võib olla raskendatud. Läbi *release*-protseduuri automatiseerimise oleks võimalik selliseid kitsaskohti vältida, teostades *clean build*'e CI serveris.

Üheks olemasoleva süsteemi probleemiks on ka see, et perioodilise automaatse ehitamise puudumise korral võib tekkida olukord, kus mõni hooldusfaasis olev projekt ühtäkki enam ei kompileeru ning on keeruline mõista, miks ja millal projekt kompileerumisvõime kaotas. Põhjuseid selleks võib olla mitmeid, näiteks mõne teegi uuenemine arendajate

töömasinates kasutatava operatsioonisüsteemi tarkvaravaramutes. Eriti sagedasti lõpetavad vanemad projektid kompileerumise siis, kui uuendatakse tervet operatsioonisüsteemi – sellega kaasneb tavaliselt ka C standardteegi (ettevõtte Linux Mint põhistes tööjaamades *glibc*) uuendamine, mis ei garanteeri ehitusprotsessi vaatest tagasiühilduvust.

Alljärgnevalt on välja toodud eelmainitud põhilised olemasoleva süsteemi probleemid nummerdatuna. Järgnevates töö peatükkides viidatakse nendele probleemidele numbriliselt.

P1 Koodimuudatuste integreerimise harv sagedus.

P2 Aeganõudvad koodi läbivaatused.

P3 Ajakulukas manuaalne testimine.

P4 Aja- ja ressursikulukad *clean build*'id.

P5 Hooldusfaasis olevad projektid, mis kaotavad ajapikku kompileerumisvõime.

3.4 Olemasoleva süsteemi probleemide analüüs

Ülevaade olemasoleva süsteemi probleemidest on kirjutatud üldistavas võtmes. Tulenevalt ettevõtte arendusprotsessi projektipõhisest loomust on iga projekt erinev ning välja toodud probleemide kaalud muutuvad vastavalt sellele. Näiteks väiksemate projektide puhul, mida võib arendada ainult üks arendaja, ei ole probleem P1 asjakohane. Kui projekt on koodimahult väike, siis pole aktuaalne probleem P4.

Olemasoleva süsteemi probleemid jagunevad kaheks:

- probleemid, mille lahendamisel oleks CI süsteemi kasutuselevõtul toetav roll – probleemid P1–P3;
- probleemid, mille CI süsteemi kasutuselevõtt lahendaks otseselt – probleemid P4 ja P5.

Vajaduse sellise eristuse järgi tingib asjaolu, et probleemid P1–P3 ei ole lahendatavad puhtalt CI süsteemi kasutuselevõtuga ning nõuavad täiendavaid organisatsioonisiseseid muudatusi.

Probleemi P1 lahendamine nõuab muutust ettevõtte arenduskultuuris – arendajaid tuleb innustada tegema sagedasi, mahult väikseid tõmbetaotlusi. See muudab koodi läbivaatused lihtsamaks ja kahandab tõenäosust integratsioonikonfliktide tekkeks. CI süsteemil saab selle probleemi lahendamisel olla vaid toetav roll – võib loota, et CI süsteemiga kaasnevad teised muudatused (automatiseerimine, automaattestide kasutamine) julgustavad ja distsiplineerivad arendajaid süsteemist maksimumi võtma, ehk siis võimaldama koodimuudatuste võimalikult sagedast integreerimist.

Probleemi P2 lahendamisele saab CI süsteem kaasa aidata läbi läbivaataja töö lihtsustamise. Tehes läbivaatajale mugavalt kättesaadavaks kogu CI teadaolev info tõmbetaotlusele vastavast koodiseisust saab läbivaataja teha oma otsuse sellele toetudes. Selle info hulka võiks kuuluda näiteks:

- Kas läbivaadatav koodiseis kompileerub ning milliseid (uusi) kompilaatorihoiatusi esines?
- Kas läbivaadatav koodiseis vastab ettevõttesisestele koodivormindusreeglitele?
- Milliseid teateid andsid kasutatavad koodianalüüsi tööriistad?
- Kas kõik automaattestid õnnestusid?

Kui CI süsteemi poolt agregeeritud vastused kõikidele ülaltoodud küsimustele viitavad sellele, et kood on korrektne, võib läbivaataja teha oma otsuse oluliselt suurema enesekindlusega ning saab keskenduda koodi funktsionaalse korrektsuse kontrollimisele.

Probleemi P3 lahendamisel on CI süsteemil osaliselt toetav roll. Põhiline rõhk lasub arenduspraktikate muutmisel – tuleb juurutada automaattestide kirjutamise praktikat. Sealjuures tasub arvestada, et testitava koodi kirjutamine on keeruline ning nõuab arendajatelt pingutust ja distsipliini. Probleemi lahendamisele aitab kaasa asjaolu, et kord juba valmis

kirjutatud automaattestid on kasutatavad ka edaspidi ning automaattestide pidev käivitamine CI lahenduse poolt vähendab manuaalsele testimisele kuluvat aega. Põhjalike automaattestide olemasolul saab arendaja pärast koodimuudatuste tegemist olla suhteliselt veendunud, et uued muudatused ei põhjustanud mõne vana funktsionaalsuse katkiminekut – pole vaja peale iga muudatust olemasolevaid funktsionaalsuseid manuaalselt testida. Ühtlasi võib ka arvata, et CI süsteemi kasutuselevõtmine ärgitab arendajaid testimisele suuremat tähelepanu pöörama.

Probleemid P4 ja P5 võimaldab CI süsteemi kasutuselevõtt otseselt läbi vastavate tegevuste automatiseerimise lahendada.

Kokkuvõtteks peab väljapakutav arendusprotsess koos kasutusele võetava CI süsteemiga lahendama kõik välja toodud probleemid.

4 CI protsessi kaardistamine

Peatükis defineeritakse kriteeriumid, millele valitav CI lahendus peab vastama. Võrdluseks eelmises peatükis kirjeldatud AS-IS protsessile kirjeldatakse ära, milline võiks välja näha ettevõtte arendusprotsess pärast valitava CI lahenduse juurutamist – töötatakse välja TO-BE protsess.

4.1 Kriteeriumid, millele CI lahendus peab vastama

Järgnevalt kirjeldatakse kriteeriume, millele valitav CI lahendus vastama peab. Järgnevates töö osades viidatakse nendele kriteeriumitele numbriliselt.

K1 Lahendus peab olema võimalikult universaalne. Lahendus peab toetama kõiki tarkvaraprojekte, sõltumata nende kasutatavast ehitussüsteemist, programmeerimiskeelest või kompilaatorist. Lahendus peab võimaldama kergesti välja vahetada keskkondi, kus CI konveiereid käivitatakse, näiteks virtualiseerimise abil.

K2 Lahendus peab sisaldama integreerimisvõimalust ettevõttes hetkel kasutusel oleva koodivaramuplatvormiga. Integreerimisvõimaluste alla võivad kuuluda näiteks järgnevad võimalused:

- Erinevate CI protsesside käivitamine kindlate sündmuste tagajärjel (näiteks projekti ehitusprotsessi käivitamine koodimuudatuste lisamise või tõmbetaotluse tekitamise tagajärjel).
- Konveierite seadistamine, käivitamine ja nende tulemuste vaatamine otse läbi koodivaramuplatvormi kasutajaliidese.

K3 Lahendus peab olema kuluefektiivne – lahenduse kasutamine ei tohi olla majanduslikult kulukam kui on eeldatav lahenduse kasutamisest saadav tulu.

K4 Lahendus peab võimaldama HIL-testimist.

K5 Lahendus peab olema lihtsasti kasutatav ja hästi dokumenteeritud.

4.2 TO-BE CI protsessi välja töötamine

Järgnevalt kirjeldatakse tüüpilist tarkvaraaarendusprotsessi arendaja vaatest sellisena, nagu see pärast planeeritud muudatusi võiks olla. Protsessi sisendiks on arendajale antud tööülesanne ning vastava projekti jaoks seadistatud CI lahenduse konveierid. Esimene konveier (edaspidi ehituskonveier) sisaldab juhendit, kuidas käib konkreetse projekti kompileerimine, kuidas käivitada koodianalüüsi- ja vormindustööriistad ja automaattestid, ning kuidas talletada ja interpreteerida nende väljundit. Teine konveier (edaspidi väljalaskekonveier) sisaldab juhendit väljalaskeprotseduuri kohta – kuidas nimetada saadusfailid, kas ja kuidas genereeritakse muudatuste logi, ning millisesse failiserveri kausta need liigutatakse.

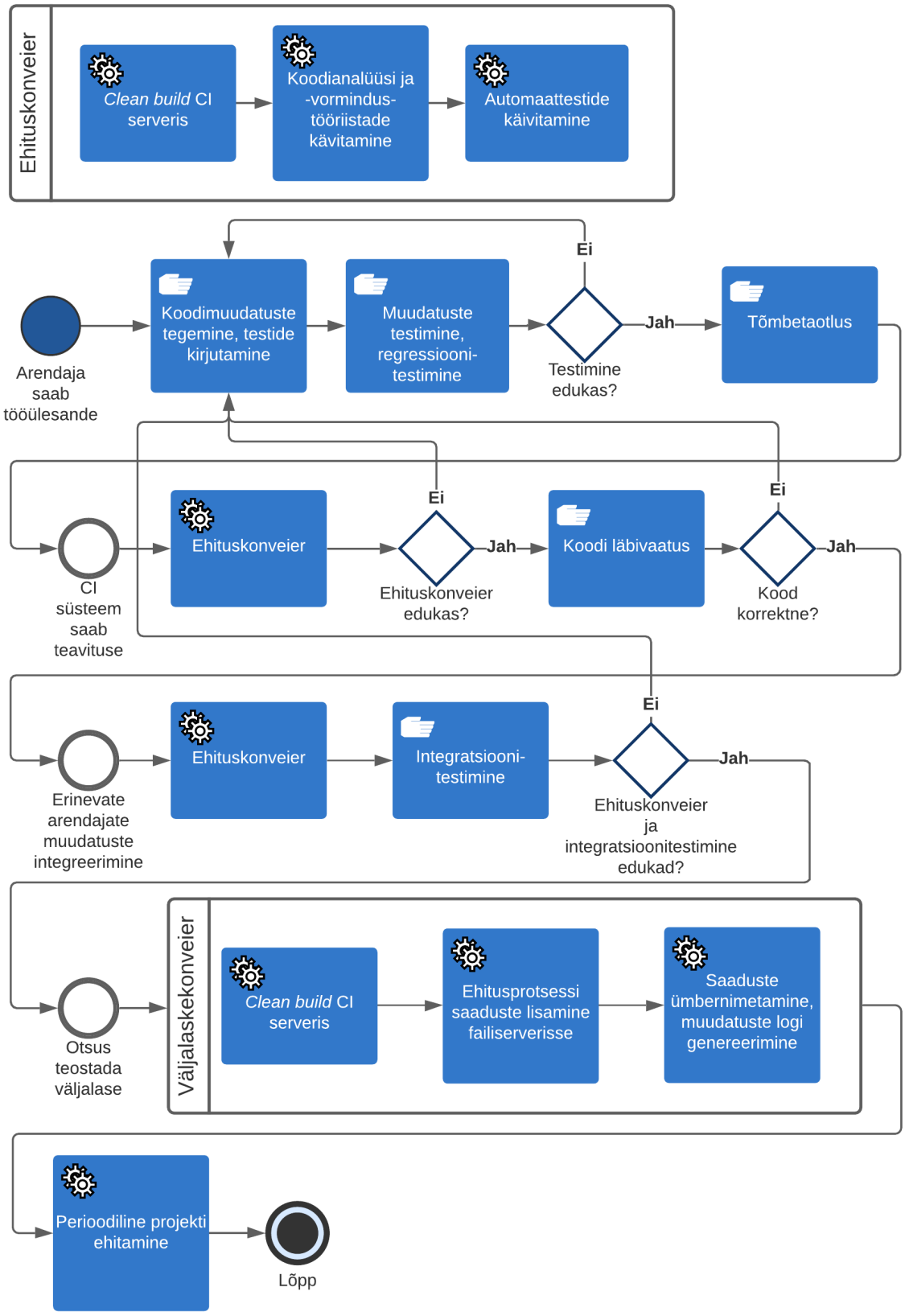
Arendaja viib läbi koodimuudatusi, teostades arendustööd n-ö isiklikus koodihoidla harus. Arendaja kirjutab teste lisatavate funktsionaalsete nõuete täitmiseks mõeldud koodiosa verifitseerimiseks ning pöörab üldist tähelepanu testitava koodi kirjutamisele. Koodimuudatuste tegemisel paneb arendaja rõhku sellele, et muudatustele vastavad *commit*'id ning tõmbetaotlused oleksid võimalikult väikesemahulised ning konkreetsed. Antud praktikate järgimine (koos efektiivsete ning kiirete koodi läbivaatustega) lahendab probleemi P1. Olles koodimuudatuste tegemise lõpetanud, testib arendaja tehtud muudatusi nii riistvaral kui ka automaattestide abil. Automaattestide kasutamine aitab vältida regressioonivigade tekkimist ning vähendab manuaalsele testimisele kuluvat aega, aidates kaasa probleemi P3 lahendamisele. Soovituslikult käivitab arendaja enne koodi *commit*'imist ka projekti juures kasutatavad koodianalüüsi- ja vormindustööriistad, et koodi korrektsuses veenduda. Arendaja laeb muudatused eraldi haruna Azure DevOps koodivaramusse ning teostab tõmbetaotluse koodihoidla peaharusse. Seejärel teavitatakse sellest CI serverit, mispeale käivitatakse ehituskonveier, mis projekti puhtast seisust kokku kompileerib, talletades esinevad kompilaatori veateated ja hoiatused. Pärast tarkvara ehitamist kontrollitakse lähtekoodi vastavust ettevõtte standardsetele koodivormindamisreeglitele ning käivitatakse koodianalüüsi tööriistad ning automaattestid. Seejärel agregeeritakse tulemused kergesti loetavale kujule, misjärel on igal vastava projektiga seotud arendajal nendele juurdepääs. Enne tõmbetaotluse üle vaatamist tutvub läbivaataja mainitud tulemustega.

Kui mõni kontrollitavatest aspektidest andis mitterahuldava tulemuse (kood ei vastanud vormindamisreeglitele, koodianalüüsi tööriistad tuvastasid olulise vea või ebaõnnestusid automaattestid), siis lükkab läbivaataja tõmbetaotluse tagasi ning arendaja viib läbi muudatused, et esinenud vead parandada. Kui CI serveri poolt läbi viidud sammud viitavad sellele, et kood on korrektne, saab läbivaataja teha oma otsuse oluliselt suurema enesekindlusega ning saab keskenduda koodi funktsionaalse korrektsuse kontrollimisele. Sellised sammud loovad soodsa keskkonna probleemi P2 lahendamiseks. Pärast tõmbetaotluse aktsepteerimist ning koodimuudatuste peaharusse mestimist saab CI server selle kohta jällegi teavituse, mispeale teostatakse jällegi *clean build* ning kontrollitakse koodi korrektsust. Erinevusena eelnevatest sammudest talletatakse peaharust ehitatud saadused.

Peale väljalaske teostamise otsust on CI serveris juba olemas vastava *commit*'iga seotud *clean build*'i tulemusel tekkinud saadusfailid. Seega piisab väljalaske teostamiseks vaid CI lahendusele vastava käsu andmisest, mispeale käivitatakse väljalaskekonveier. Vajadusel nimetab konveier ümber saadusfailid ning genereerib muudatuste logi, kasutades selleks versioonihaldussüsteemi *commit message*'id. Seejärel liigutab konveier vastavalt eelnevalt defineeritud reeglitele saadusfailid ja muudatuste logi automaatselt failiserverisse, *Release* kausta. Selline väljalaskeprotseduur lahendab probleemi P4 ning jätab võrreldes AS-IS protsessiga vähem ruumi inimvigade tekkeks.

Järgneb ehituskonveieri perioodiline käivitamine. Perioodiline projekti ehitamine toimub vastavalt peaharu hetkeseisule ning selle eesmärgiks on lahendada probleem P5.

Protsessist ülevaate saamiseks on loodud töövoog joonisel 2.



Joonis 2. TO-BE protsessi töövoog.

5 Sobiva CI lahenduse analüüs ja valimine

Peatükis kirjeldatakse ära metoodika, mille alusel valitakse ettevõttele sobivaim CI lahendus. Seejärel analüüsitakse eelnevalt välja toodud CI lahendusi ning valitakse nende seast sobivaim.

5.1 Metoodika

Sobivaima CI lahenduse valimiseks otsutati kasutada otsustusmaatriksi meetodit. Selle rakendamiseks pannakse järgnevalt paika hindamisskaala, mis kirjeldab peatükis 4.1 kirjeldatud kriteeriumeid 0–5 punkti vahemikus. Konkreetsemate kriteeriumite puhul ei kasutata skaalat täies ulatuses – vastavaid kriteeriume hinnatakse kas 0, 3 või 5 punktiga. Kui mõne kriteeriumi puhul on võimalik anda erinevaid hinnanguid (näiteks on osad lahendused saadaval nii pilv- kui ka isemajutatud variandis), siis valitakse vaikimisi võimalus, mis annab kõrgeima punktisumma. Hindamisskaala on välja toodud tabelites 2–6. Valiku objektiivsuse tagamiseks omistatakse vaatlusalustele kriteeriumitele nende tähtsust tähistavad kaalud. Kaalude leidmiseks viiakse ettevõtte võtmeisikute seas läbi küsitlus, milles kasutatakse erinevatele kriteeriumitele tähtsushinnangute andmiseks MoSCoW meetodit[1]. Seejärel teisendatakse tulemused numbrilisele skaalale. Teisendamisprotsessi kirjeldab tabel 1. Lõplike kaalude saamiseks arvutatakse kõigi võtmeisikute vastuste põhjal iga kriteeriumi kohta välja vastuste geomeetriline keskmine. Geomeetrilise keskmise kasutamise kasuks otsustati, kuna see annab ebasümmeetrilise andmestiku korral vähem moonutatud tulemuse, kui näiteks aritmeetiline keskmine või mediaan [27]. Pärast kaalude korrutamist erinevate lahenduste saadud punktidega summeeritakse punktid. Valituks osutub lahendus, mis saab kaalutud hinnangute summerimisel kõrgeima punktisumma.

[1] `https://www.agilebusiness.org/page/ProjectFramework_10_MoSCoWPrioritisation`

Tabel 1. MoSCoW meetodi tulemuste teisendamine numbrilisele skaalale.

MoSCoW hinnang	**Numbriline hinnang**
Must have	4
Should have	3
Could have	2
Won't have	1

Tabel 2. Kriteerium K1 hindamisskaala.

Punkte	**Kirjeldus**
0	Lahendus toetab ainult kindlat hulka programmeerimiskeeli, ehitussüsteeme või kompilaatoreid.
3	Lahendus toetab mistahes programmeerimiskeelte, ehitussüsteemide või kompilaatorite kasutamist.
5	Lahendus toetab mistahes programmeerimiskeelte, ehitussüsteemide või kompilaatorite kasutamist. Lahendusel on sisseehitatud tugi konveierite käivitamiseks virtualiseeritud keskkondades.

Tabel 3. Kriteerium K2 hindamisskaala.

Punkte	**Kirjeldus**
0	Lahenduse integreerimine olemasoleva koodivaramuplatvormiga pole võimalik. Kõikide lahendusega seonduvate toimingute käivitamine peab toimuma manuaalselt.
3	Lahendus toetab erinevate võimalike sündmuste (koodimuudatuste *push*, tõmbetaotluse tekitamine) haakimist erinevate tegevuste automatiseerimisel.
5	Lahendus integreerub olemasoleva koodivaramuplatvormiga täielikult. CI lahenduse enimkasutatav funktsionaalsus (konveierite konfigureerimine, käivitamine ja nende logide vaatamine, ehitusprotsesside saadustele ligi pääsemine, konveierite tulemuste agregeeritud kuju vaatamine) on saavutatav otse läbi olemasoleva koodivaramuplatvormi. Lahendus võimaldab erinevate CI protsesside automaatset käivitamist kindlate sündmuste (koodimuudatuste *push*, tõmbetaotluse tekitamine) tagajärjel.

Tabel 4. Kriteerium K3 hindamisskaala.

Punkte	**Kirjeldus**
0	Lahendus on isemajutatud ning sellega kaasnevad lahenduse haldamisega seotud kulud (tööjõukulud, infrastruktuuri kulud). Lahenduse kasutamisega kaasnevad eeldatavasti kõrged jooksvad kulud.
1	Vajadus lahenduse haldamiseks puudub kuna lahenduse majutamisega tegeleb kolmas osapool. Lahenduse kasutamisega kaasnevad eeldatavasti kõrged jooksvad kulud.
2	Lahendus on isemajutatud ning sellega kaasnevad lahenduse haldamisega seotud kulud (tööjõukulud, infrastruktuuri kulud). Lahenduse kasutamisega kaasnevad eeldatavasti madalad jooksvad kulud.
3	Vajadus lahenduse haldamiseks puudub, kuna lahenduse majutamisega tegeleb kolmas osapool. Lahenduse kasutamisega kaasnevad eeldatavasti madalad jooksvad kulud.
4	Lahendus on isemajutatud ning sellega kaasnevad lahenduse haldamisega seotud kulud (tööjõukulud, infrastruktuuri kulud). Lahenduse kasutamine ei too endaga peale majutuskulude kaasa täiendavaid kulusid.
5	Vajadus lahenduse haldamiseks puudub, kuna lahenduse majutamisega tegeleb kolmas osapool. Lahenduse kasutamine ei too endaga kaasa täiendavaid kulusid.

Tabel 5. Kriteerium K4 hindamisskaala.

Punkte	**Kirjeldus**
0	Lahendus ei võimalda HIL-testimist. Lahendusel puudub API, mille abil oleks võimalik liidestada suhtlust riistvaraga.
3	Lahendusel puudub otsene tugi HIL-testimiseks, kuid seda on võimalik teostada läbi lahenduse API.
5	Lahendusel on otsene tugi HIL-testimiseks.

Tabel 6. Kriteerium K5 hindamisskaala.

Punkte	**Kirjeldus**
0	Lahendusel puudub dokumentatsioon.
1	Lahenduse dokumentatsioon on pealiskaudne ning näited enamsooritatavate tegevuste kohta puuduvad.
2	Lahenduse dokumentatsioon on rahuldav. Leidub näiteid osade enamsooritavate tegevuste kohta.
3	Lahenduse dokumentatsioon on põhjalik ning sisaldab näiteid kõigi enamsooritatavate tegevuste kohta.
4	Lahenduse dokumentatsioon on põhjalik ning sisaldab näiteid kõigi enamsooritatavate tegevuste kohta. Saadaval on mitteametlik dokumentatsioon erinevate sardsüsteemide tarkvarale spetsiifiliste näidetega.
5	Lahenduse dokumentatsioon on põhjalik ning sisaldab näiteid kõigi enamsooritatavate tegevuste kohta. Saadaval on ametlik dokumentatsioon erinevate sardsüsteemide tarkvarale sptesiifiliste näidetega.

5.2 Lahenduste analüüs

Järgnevalt on hinnatud iga vaadeldavat lahendust vastavalt peatükis 5.1 välja toodud metoodikale.

Jenkins

Tabel 7. Hinnangud Jenkins CI lahendusele.

Kriteerium	**Hinnang**	**Kirjeldus**
K1	5	Jenkins toetab mistahes projektide ehitamist ja automatiseerimist [28]. Jenkinsi konveierite abil on võimalik Groovy programmerimiskeelel põhinevas valdkonnaspetsiifilises keeles kirjeldada mistahes automatiseerimisprotsesside sooritamiseks vajaminevaid samme – võimalik on läbi operatsioonisüsteemi kesta (ingl k *shell*) välja kutsuda mistahes paigaldatud programme [14]. Lisaks sellele on võimalik konveiereid jooksutada virtualiseeritud režiimis Docker konteinerites [29].
K2	3	Azure DevOps platvorm toetab erinevate koodivaramus toimuvate sündmuste haakimist Jenkinsiga [30].
K3	4	Jenkins on täielikult tasuta kasutatav tarkvara ning ei paku ametlikku majutusteenust [12].
K4	3	Jenkins võimaldab HIL-testimist tänu võimalusele käivitada konveierites mistahes programme [14].
K5	5	Jenkinsi dokumentatsioon on mahukas ja põhjalik ning sisaldab hulgaliselt näiteid. Jenkinsi kasutamisest sardsüsteemide tarkvaraarenduse valdkonnas leidub nii ametlikku [31], kui ka mitteametlikku [32] dokumentatsiooni.

TeamCity

Tabel 8. Hinnangud TeamCity CI lahendusele.

Kriteerium	**Hinnang**	**Kirjeldus**
K1	5	TeamCity toetab mistahes projektide ehitamist ja automatiseerimist. TeamCity konveierite abil on võimalik vabas vormis Kotlin programmeerimiskeeles kirjeldada mistahes automatiseerimisprotsesside sooritamiseks vajaminevaid samme – võimalik on läbi operatsioonisüsteemi kesta välja kutsuda mistahes paigaldatud programme. [33] Lisaks sellele on võimalik konveiereid jooksutada virtualiseeritud režiimis Dockeri konteinerites [34].
K2	3	TeamCity võimaldab läbi HTTP API liidestust Azure DevOps platvormiga [35].
K3	0	TeamCity on saadaval JetBrainsi poolt majutatud variandis TeamCity Cloud [36]. Kuna hetkel on antud teenus veel beetastaadiumis, siis seda valikut ei kaaluta. TeamCity on saadaval tasuta variandina, kuid sel juhul on ehitusagentide arv piiratud kolmele ning erinevate ehituskonfiguratsioonide arv piiratud sajale. Uurimisaluse ettevõtte kontekstis jääb sajast ehituskonfiguratsioonist üsna pea ilmselt väheks. Seepeale tuleb neid juurde osta; 10 ehituskonfiguratsiooni ja 1 ehitusagent maksab aastas 299€. Alternatiivne variant on osta kolme ehitusagendiga Enterprise Server litsents, mis maksab 1999€ aastas ja eemaldab ehituskonfiguratsioonide piirangu täielikult. Edasiste ehitusagentide lisamisel kasvab hind veelgi. [17]
K4	3	TeamCity võimaldab HIL-testimist tänu võimalusele käivitada konveierites mistahes programme [37].
K5	3	TeamCity dokumentatsioon on mahukas ja põhjalik ning sisaldab hulgaliselt näiteid. Sardsüsteemide tarkvaraarendusele spetsiifilist dokumentatsiooni ei leitud.

BuildBot

Tabel 9. Hinnangud BuildBot CI lahendusele.

Kriteerium	**Hinnang**	**Kirjeldus**
K1	3	BuildBot lahenduse konfigureerimine toimub läbi Pythoni skriptide, mille abil on võimalik kirjeldada mistahes automatiseerimisprotsesse [38].
K2	3	BuildBot võimaldab liidestust Azure DevOps platvormiga läbi HTTP API [39].
K3	4	BuildBot on tasuta kasutatav tarkvara, ametlikku majutusteenust ei eksisteeri [40].
K4	3	BuildBot võimaldab HIL-testimist tänu võimalusele käivitada konveierites mistahes programme [38].
K5	3	BuildBot dokumentatsioon on mahukas ja põhjalik ning sisaldab hulgaliselt näiteid. Sardsüsteemide tarkvaraarendusele spetsiifilist dokumentatsiooni ei leitud.

Gitlab CI/CD

Tabel 10. Hinnangud GitLab CI/CD lahendusele.

Kriteerium	**Hinnang**	**Kirjeldus**
K1	5	GitLab CI/CD konveierite konfigureerimine toimub YAML serialiseerimiskeele abil. Tänu võimalusele käivitada konveierites mistahes programme toetab GitLab CI/CD mistahes projektide ehitamist ja automatiseerimist. Toetatud on ka konveierite käivitamine virtualiseeritud režiimis Dockeri konteinerites. [41]
K2	3	GitLab CI/CD võimaldab liidestust Azure DevOps platvormiga läbi HTTP API [42].
K3	1	Kuigi GitLabi on võimalik kasutada tasuta, siis GitLab CI/CD kasutamine koos välise koodivaramuplatvormiga on võimalik ainult Premium ja Ultimate litsentsiplaanides [43]. Need maksavad iga kasutaja kohta vastavalt $19 ja $99 kuus. Mõlemad litsentsiplaanid sisaldavad majutusvõimalust, seega pole olulist põhjust isemajutatud versiooni kasutamiseks. Antud plaanid võimaldavad jooksutada CI konveiereid vastvalt 10 000 ja 50 000 minutit kuus. Mõlema plaani puhul on võimalik CI töölisservereid ka ise majutada – isemajutatud serveritel CI konveierite jooksutamise minutite piirang puudub. [23]
K4	3	GitLab CI/CD võimaldab HIL-testimist tänu võimalusele käivitada konveierites mistahes programme [41].
K5	4	GitLab CI/CD dokumentatsioon on mahukas ja põhjalik ning sisaldab hulgaliselt näiteid. Leidub sardsüsteemide tarkvaraarendusele spetsiifilist mitteametlikku dokumentatsiooni. [44], [45]

Azure Pipelines

Tabel 11. Hinnangud Azure Pipelines CI lahendusele.

Kriteerium	**Hinnang**	**Kirjeldus**
K1	5	Azure Pipelines lahendus võimaldab automatiseerida mistahes projekte [24]. Lahendus lubab käivitada konveierites mistahes programme ning toetab konveierite jooksutamist virtualiseeritud režiimis Dockeri konteinerites [46].
K2	5	Olles osa Azure DevOps platvormist, integreerub Azure Pipelines ettevõtte koodivaramuplatvormiga täielikult.
K3	4	Azure Pipelines kasutamine on tasuta, kuni ühes kuus kasutatakse vähem kui 1800 minuti jagu konveierite tööaega [47]. Edaspidi tuleb iga järgneva 1800 minuti eest kuus maksta $40 [48]. Alternatiivina on võimalik ehitusagente ise majutada, nendel agentidel ajapiirangud puuduvad, kuid isemajutatatavate agentide arv on piiratud vastavalt ettevõtte Visual Studio Enterprise teenuse tellijate arvule [47]. Uurimisaluses ettevõttes on antud teenuse tellijaks iga arendustiimi liige. Kuna ettevõtte arendusprojektid on tihti koodimahult suured ning vajaminevate ehitusminutite arvu on raske hinnata (sõltub projektist ning riistvara võimekusest), siis otsutati, et kuluefektiivsem on ehitusagentide isemajutamine.
K4	3	Azure Pipelines võimaldab HIL-testimist tänu võimalusele käivitada konveierites mistahes programme [25].
K5	3	Azure Pipelines dokumentatsioon on mahukas ja põhjalik ning sisaldab hulgaliselt näiteid. Sardsüsteemide tarkvaraarendusele spetsiifilist dokumentatsiooni ei leitud.

5.3 Sobiva lahenduse valimine

Kriteeriumite tähtsuste hindamiseks viidi ettevõtte võtmeisikute seas läbi küsitlus, mis on välja toodud lisas 2. Küsitlus esitati vastamiseks kolmele ettevõtte töötajale, kelle rollid ettevõttes on järgmised:

- tarkvaraarenduse juht;
- tarkvaraarenduse projektijuht;
- vaneminsener.

Tabelis 12 on välja toodud küsitluse tulemused, mis on eelnevalt vastavalt tabelile 1 teisendatud numbrilise kujule; välja on toodud ka iga kriteeriumi kohta käivate tulemuste geomeetriline keskmine ehk lõplik kaal, mille abil valik tehakse.

Tulemused on ootuspärased ning kooskõlas autori hinnangutega. Nagu näha peeti ühtlaselt kõige olulisemateks kriteeriumiteks lahenduse universaalsust (K1) ning integratsioonivõimalust (K2) olemasoleva koodivaramuplatvormiga. Olulisuselt järgnesid lahenduse kuluefektiivsus (K3) ja võimalust kasutusele võtta HIL-testimine (K4). Madalaima tähtsusega kriteeriumiks hinnati dokumentatsiooni olemasolu ning lahenduse kasutamise lihtsust (K5). Ettevõttes, kus on suur insenerikompetents, pole dokumentatsiooni olemasolu üleliia oluline. Samas on tänu ettevõttes kasutatavate tehnoloogiate mitmekesisusele väga oluline, et lahendus oleks võimalikult universaalne.

Tabel 12. Küsitluse tulemused.

Kriteerium / **Vastaja roll**	K1	K2	K3	K4	K5
Tarkvaraarenduse juht	4	4	4	3	3
Tarkvaraarenduse projektijuht	4	4	2	3	2
Tarkvarainsener	4	4	4	3	3
Geomeetriline keskmine	4	4	3.17	3	2.62

Tabelis 13 on välja toodud iga lahenduse kaalutud punktihinnangud iga kriteeriumi eest. Hinnanguteni jõuti, korrutades iga lahenduse saadud punkthinnang läbi vastava kriteeriumi kaaluga. Täpsem arvutuskäik on välja toodud lisas 3. Nagu näha, saavutas suurima punktisumma lahendus Azure Pipelines. Märkimist tasub asjaolu, et kui uurimisalusel ettevõttel poleks kasutusel Azure DevOps koodihaldusplatvorm, oleks valituks osutunud hetkel teisele kohale jäänud lahendus Jenkins – Azure Pipelines sai kõrgeima skoori eelkõige tänu kriteeriumile K2. Tulemus on ootuspärane ning kooskõlas autori hinnanguga.

Tabel 13. Lahendustele kaalutud hinnangute andmine.

CI lahendus / **Kriteerium**	**Jenkins**	**TeamCity**	**BuildBot**	**GitLab CI/CD**	**Azure Pipelines**
K1	20	20	12	20	20
K2	12	12	12	12	20
K3	12.68	0	12.68	3.17	12.68
K4	9	9	9	9	9
K5	13.1	7.86	7.86	10.48	7.86
Kokku	66.78	48.86	53.54	54.65	69.54

5.4 Lahenduse testimine

Et ka praktikas lahenduse kasutamise võimalikkuses veenduda, otsustati selle abil luua ühele ettevõtte tarkvaraprojektile CI ehituskonveier. Valitud projektiks osutus STM32 perekonna mikrokontrolleri püsivara, mis on kirjutatud C programmeerimiskeeles, ning mille ehitussüsteemiks on CMake. Projekti kompileeritakse Linaro GCC krosskompilaatoriga. Tegemist on ettevõttes suhteliselt tavapärase projektiga, tänu millele peaks välja töötatud konveier olema lihtsasti kohandatav mitmetele teistele ettevõtte projektidele.

Ehituskeskkonna seadistamiseks otsustati kasutada Dockeri süsteemipilti. Dockeri abil on võimalik täpselt defineerida keskkond, kus konveieri samme käivitama hakatakse, kusjuures keskkonna seadistamine ning konveieri sammud on üksteisest loogiliselt eraldatud. Alternatiivina Dockeri kasutamisele oleks võimalik ehituskeskkond püsti panna ka konveieri sees, kuid Dockeri kasutamisel on mitmeid eeliseid. Olulise näitena võib välja tuua süsteemipiltide korduvkasutatavuse – sama süsteemipilti on võimalik kasutada ka teiste projektide juures. Võimalik on ka järgnevate süsteemipiltide defineerimisel olemasolevaid ära kasutada, võttes olemasolev süsteemipilt uue põhjaks.

Süsteemipildi loomisel võetakse põhjaks Ubuntu 20.04 operatsioonisüsteem, mis on saadaval olevate tarkvarapakettide ja keskkonna poolest väga sarnane ettevõtte tarkvaraarendajate poolt kasutatava Linux Mint 20 operatsioonisüsteemiga. Seejärel paigaldatakse tarkvarapaketid, mis on vajalikud kas projekti ehitamiseks, CI konveieri käivitamiseks või skripti järgnevate sammude täitmiseks. Viimase sammuna lisatakse süsteemipilti ettevõttes enimkasutatavad krosskompilaatorid. Dockeri süsteemipildi loomise skript asub lisas 4.

Ehituskonveieri skript on defineeritud kahe eraldiseisva sammuna. Esimeses kontrollitakse koodi vastavust ettevõtte koodivormindusreeglitele Clang-Format tööriista abil. Reeglid on defineeritud koodivaramu juurkaustas asusvas failis, mille kasutatav tööriist ise üles leiab. Teise sammu käigus kompileeritakse projekt ning käivitatakse automaattestid. Kasutatavaks testimisraamistikuks on GoogleTest (gtest). Testide käivitamisel väljastab raamistik tulemused standardiseeritud formaadis ka XML-faili, mille abil Azure Pipelines

tulemused koondab. Viimase sammuna avaldatakse allalaadimiseks ehitamise tulemusena tekkinud püsivarapaketid. Testiti ka konveieri automaatset käivitamist koodimuudatuste tegemise või tõmbetaotluse tekitamise tagajärjel – mõlemad võimalused toimisid ootuspäraselt. Ehituskonveieri skript asub lisas 5.

Ehituskonveieri skripti valmimise tulemusel hinnati valitud lahenduse pakutav funktsionaalsus piisavaks, et ettevõtte vajadused saaksid rahuldatud.

6 Võimalikud edasiarendused

Ühe vajaliku ja võimaliku edasiarendusena näeb autor välja töötatud protsessi ning valitud lahenduse testimist ettevõtte mõne reaalse projekti juures. Testimine võiks muuhulgas hõlmata arendustsüklite pikkuse mõõtmist – näiteks saaks mõõta väljalaskeprotseduurile kuluvat aega, sooritades see nii manuaalselt, ehk vastavalt AS-IS protsessile, kui ka automatiseeritult, ehk vastavalt TO-BE protseduurile. Läbi protsessi ja lahenduse testimise avalduksid ilmselt nii mõlema puudujäägid kui ka järgnevad võimalikud edasiarendused; samuti oleks võimalik testimise tulemuste järgi paika panna protsessi ja lahenduse täpsem juurutamisplaan.

Teise võimaliku edasiarendusena näeb autor valitud lahenduse ja välja töötatud protsessi sidumist HIL-testimisega. HIL-testimine on kahtlemata väga suurt potentsiaali omav töövahend, mille abil oleks võimalik jõuda sardsüsteemide tarkvaraarenduse valdkonnas lähemale mõtteviisile, mis on omane eelkõige teistele tarkvaraarenduse valdkondadele – "kui tarkvaraprojekt läbib edukalt kõik testid, siis see ka töötab". Tänu väga tugevale sõltuvusele riistvarast on sellist väidet sardsüsteemide arendamise kontekstis praktiliselt võimatu esitada, kuid põhjalik HIL-testimine oleks siiski suur samm õiges suunas.

7 Kokkuvõte

Käesoleva bakalaureusetöö käigus uuriti pidevintegratsiooni kasutamist ühes sardsüsteemide tarkvaraarendusega tegelevas Eesti tootearendusettevõttes. Töö eesmärgiks oli uurida CI rakendamist sardsüsteemide tarkvaraarenduse kontekstis, kavandada uurimisalusele ettevõttele tarkvaraarenduse protsess, mis hõlmaks endas CI praktikaid, ning valida turul pakutavate CI lahenduste seast ettevõttele sobivaim.

CI protsessi kavandamise käigus tutvustati olemasolevat arendusprotsessi, analüüsiti selle kitsaskohti ja puuduseid ning pakuti neile lahendused. Töötati välja uus arendusprotsess, mis lahendaks eelnevalt paika pandud probleemid ning hõlmaks endas CI praktikaid.

Ettevõttele sobivaima CI lahenduse valimiseks viidi läbi eelanalüüs, mille tulemusena lisati valimisse viis turul pakutavat lahendust. Seejärel pandi paika kriteeriumid, mille järgi lahendusi hinnata. Kriteeriumite tähtsuse väljaselgitamiseks viidi ettevõtte võtmeisikute seas läbi küsitlus. Valiku tegemiseks kasutati otsustusmaatriksi meetodit. Pärast analüüsi teostamist osutus valituks lahendus Azure Pipelines. Valiku tulemus oli ootuspärane ning kooskõlas autori hinnanguga. Lahenduse praktilise kasutatavuse hindamiseks töötati valmis ka CI konveieri näidisskript, mille tulemusena hinnati valitud lahendus ettevõttele sobivaks.

Kasutatud kirjandus

[1] Hewlett Packard Enterprise, *Agile is the new normal*, 2017. [Online]. Loetud aadressil: https://softwaretestinggenius.com/docs/4aa5-7619.pdf Kasutatud: 17.04.2021.

[2] InfoQ, *Standish Group 2015 Chaos Report - Q&A with Jennifer Lynch*. [Online]. Loetud aadressil: https://www.infoq.com/articles/standish-chaos-2015/ Kasutatud: 17.04.2021.

[3] S. Suomi, "Project Management Tools in Agile Embedded Systems Development", 2014.

[4] M. Kaisti, V. Rantala, T. Mujunen, S. Hyrynsalmi, K. Könnölä, T. Mäkilä ja T. Lehtonen, "Agile methods for embedded systems development - a literature review and a mapping study", *EURASIP Journal on Embedded Systems*, vol. 2013, no. 1, p. 15, 12.2013, ISSN: 1687-3963. DOI: 10.1186/1687-3963-2013-15. [Online]. Loetud aadressil: https://jes-eurasipjournals.springeropen.com/articles/10.1186/1687-3963-2013-15 Kasutatud: 18.04.2021.

[5] Digital.ai Software, Inc., *The 14th annual "State of Agile Development" survey*, 2020. [Online]. Loetud aadressil: https://stateofagile.com/#ufh-i-615706098-14th-annual-state-of-agile-report/7027494.

[6] M. Fowler, *Continuous Integration*. [Online]. Loetud aadressil: https://martinfowler.com/articles/continuousIntegration.html Kasutatud: 16.04.2021.

[7] J. Humble ja D. Farley, *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*, 1st edition. Upper Saddle River, NJ: Addison-Wesley Professional, 27.07.2010, 512 pp., ISBN: 978-0-321-60191-9.

[8] P. M. Duvall, S. Matyas ja A. Glover, *Continuous integration: improving software quality and reducing risk*, ser. Addison-Wesley signature series. Upper Saddle River, NJ: Addison-Wesley, 2007, 283 pp., OCLC: ocn122423674, ISBN: 978-0-321-33638-5.

[9] L. E. Lwakatare, T. Karvonen, T. Sauvola, P. Kuvaja, H. Olsson, J. Bosch ja M. Oivo, "Towards DevOps in the Embedded Systems Domain: Why is It so Hard?", 01.2016. DOI: 10.1109/HICSS.2016.671.

[10] T. O. Adu, *CI/CD for embedded software development*. [Online]. Loetud aadressil: https://www.linkedin.com/pulse/cicd-embedded-software-development-timothy-adu Kasutatud: 04.04.2021.

[11] Y. Nissenboim, *How to Choose the Right Continuous Integration Tool for Embedded Systems*, Medium, 24.06.2018. [Online]. Loetud aadressil: https://medium.com/jumperiot/how-to-choose-the-right-continuous-integration-tool-for-embedded-systems-762cdb41b5c8 Kasutatud: 04.04.2021.

[12] *Jenkins User Documentation*. [Online]. Loetud aadressil: https://www.jenkins.io/doc/index.html Kasutatud: 04.04.2021.

[13] *Project governance document*. [Online]. Loetud aadressil: https://www.jenkins.io/project/governance/index.html Kasutatud: 04.04.2021.

[14] *Pipeline*. [Online]. Loetud aadressil: https://www.jenkins.io/doc/book/pipeline/index.html Kasutatud: 04.04.2021.

[15] *Getting Started with TeamCity*. [Online]. Loetud aadressil: https://www.jetbrains.com/help/teamcity/getting-started-with-teamcity.html Kasutatud: 04.04.2021.

[16] *Supported Platforms and Environments*. [Online]. Loetud aadressil: https://www.jetbrains.com/help/teamcity/supported-platforms-and-environments.html Kasutatud: 04.04.2021.

[17] *Buy TeamCity: Pricing and Licensing, More Build Agents*, JetBrains. [Online]. Loetud aadressil: https://www.jetbrains.com/teamcity/buy/ Kasutatud: 04.04.2021.

[18] *1.1. First Run — Buildbot 3.0.2 documentation*. [Online]. Loetud aadressil: https://docs.buildbot.net/current/tutorial/firstrun.html Kasutatud: 04.04.2021.

[19] *2.1. Introduction — Buildbot 3.0.2 documentation*. [Online]. Loetud aadressil: https://docs.buildbot.net/latest/manual/introduction.html Kasutatud: 04.04.2021.

[20] *User Docs | GitLab*. [Online]. Loetud aadressil: https://docs.gitlab.com/ee/user/index.html Kasutatud: 04.04.2021.

[21] *GitLab CI/CD | GitLab*. [Online]. Loetud aadressil: https://docs.gitlab.com/ee/ci/ Kasutatud: 04.04.2021.

[22] *CI/CD pipelines | GitLab*. [Online]. Loetud aadressil: https://docs.gitlab.com/ee/ci/pipelines/index.html Kasutatud: 04.04.2021.

[23] *Pricing model*, GitLab. [Online]. Loetud aadressil: https://about.gitlab.com/company/pricing/ Kasutatud: 04.04.2021.

[24] *What is Azure Pipelines? - Azure Pipelines*. [Online]. Loetud aadressil: https://docs.microsoft.com/en-us/azure/devops/pipelines/get-started/what-is-azure-pipelines Kasutatud: 04.04.2021.

[25] *Customize your pipeline - Azure Pipelines*. [Online]. Loetud aadressil: https://docs.microsoft.com/en-us/azure/devops/pipelines/customize-pipeline Kasutatud: 05.04.2021.

[26] *Understand differences between Azure DevOps Services and Azure DevOps Server - Azure DevOps*. [Online]. Loetud aadressil: https://docs.microsoft.com/en-us/azure/devops/user-guide/about-azure-devops-services-tfs Kasutatud: 04.04.2021.

[27] J. McChesney, *Why you should summarize your data with the geometric mean*, Medium, 16.10.2020. [Online]. Loetud aadressil: https://jlmc.medium.com/understanding-three-simple-statistics-for-data-visualizations-2619dbb3677a Kasutatud: 07.05.2021.

[28] *Jenkins*. [Online]. Loetud aadressil: https:/www.jenkins.io/ Kasutatud: 16.04.2021.

[29] *Using Docker with Pipeline*. [Online]. Loetud aadressil: https:/www.jenkins.io/doc/book/pipeline/docker/ Kasutatud: 16.04.2021.

[30] *Create a service hook with Jenkins - Azure DevOps*. [Online]. Loetud aadressil: https://docs.microsoft.com/en-us/azure/devops/service-hooks/services/jenkins Kasutatud: 16.04.2021.

[31] *Jenkins in the Embedded World*. [Online]. Loetud aadressil: https:/www.jenkins.io/solutions/embedded/ Kasutatud: 16.04.2021.

[32] *Continuous integration for C/C++ embedded devices with Jenkins, Docker and Conan*. [Online]. Loetud aadressil: https://blog.conan.io/2018/04/25/Continuous-integration-for-C-C++embedded-devices-with-jenkins-docker-and-conan.html Kasutatud: 16.04.2021.

[33] *Supported Platforms and Environments | TeamCity*. [Online]. Loetud aadressil: https://www.jetbrains.com/help/teamcity/teamcity/2020.2/supported-platforms-and-environments.html Kasutatud: 19.04.2021.

[34] *Docker Wrapper | TeamCity*. [Online]. Loetud aadressil: https://www.jetbrains.com/help/teamcity/teamcity/2020.2/docker-wrapper.html Kasutatud: 24.04.2021.

[35] *About TeamCity REST API | TeamCity*. [Online]. Loetud aadressil: https://www.jetbrains.com/help/teamcity/rest/teamcity-rest-api-documentation.html Kasutatud: 19.04.2021.

[36] *TeamCity Cloud | TeamCity*. [Online]. Loetud aadressil: https://www.jetbrains.com/help/teamcity/teamcity/2020.2/teamcity-cloud.html Kasutatud: 19.04.2021.

[37] *Command Line | TeamCity*. [Online]. Loetud aadressil: https://www.jetbrains.com/help/teamcity/teamcity/2020.2/command-line.html Kasutatud: 19.04.2021.

[38] *Buildbot*. [Online]. Loetud aadressil: https://buildbot.net/ Kasutatud: 19.04.2021.

[39] *2.5.16. Change Hooks — Buildbot 3.1.0 documentation*. [Online]. Loetud aadressil: https://docs.buildbot.net/current/manual/configuration/wwwhooks.html Kasutatud: 19.04.2021.

[40] *2.2.3. Installing the code — Buildbot 3.1.0 documentation*. [Online]. Loetud aadressil: https://docs.buildbot.net/current/manual/installation/installation.html Kasutatud: 19.04.2021.

[41] *Get started with GitLab CI/CD | GitLab*. [Online]. Loetud aadressil: https://docs.gitlab.com/ee/ci/quick_start/index.html Kasutatud: 24.04.2021.

[42] *Triggering pipelines through the API | GitLab*. [Online]. Loetud aadressil: https://docs.gitlab.com/ee/ci/triggers/ Kasutatud: 24.04.2021.

[43] *GitLab CI/CD for GitHub*. [Online]. Loetud aadressil: https://about.gitlab.com/solutions/github/ Kasutatud: 24.04.2021.

[44] *DevOps for embedded (part 1) – stupid projects*. [Online]. Loetud aadressil: https://www.stupid-projects.com/devops-for-embedded-part-1/ Kasutatud: 24.04.2021.

[45] *IoT DevOps Hands-On (Day 3): GitLab CI/CD and Friends - DZone IoT*, dzone.com. [Online]. Loetud aadressil: https://dzone.com/articles/hands-on-iot-devops-day-3-gitlab-cicd Kasutatud: 24.04.2021.

[46] *Container Jobs in Azure Pipelines and TFS - Azure Pipelines*. [Online]. Loetud aadressil: https://docs.microsoft.com/en-us/azure/devops/pipelines/process/container-phases Kasutatud: 24.04.2021.

[47] *Configure and pay for parallel jobs - Azure DevOps*. [Online]. Loetud aadressil: https://docs.microsoft.com/en-us/azure/devops/pipelines/licensing/concurrent-jobs Kasutatud: 24.04.2021.

[48] *Azure DevOps Services Pricing | Microsoft Azure*. [Online]. Loetud aadressil: https://azure.microsoft.com/en-us/pricing/details/devops/azure-devops-services/ Kasutatud: 24.04.2021.

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

Mina, Rihard Soodla

1. Annan Tallinna Tehnikaülikoolile tasuta loa (lihtlitsentsi) enda loodud teose "Pidevintegratsiooni protsessi kavandamine sardsüsteemide tarkvaraarendusega tegelevas ettevõttes" mille juhendaja on German Mumma
   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.

09.05.2021

[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 – CI lahenduse kriteeriumite hindamise küsitlus

Ühe oma lõputöö osana viin läbi erinevate CI lahenduste analüüsi ning valin neist meie ettevõttele sobivaima. Olen paika pannud viis kriteeriumi, mille abil erinevaid turul pakutavaid CI lahendusi hinnata. Palun alljärgnevas tabelis hinnata CI lahenduse kriteeriume vastavalt MoSCoW hindamisskaalale.

Kriteerium	**Kirjeldus**	**Kriteeriumi tähtsus (M/S/C/W)**
K1	Lahendus peab olema võimalikult universaalne. Lahendus peab toetama kõiki tarkvaraprojekte, sõltumata nende kasutatavast ehitussüsteemist, programmeerimiskeelest või kompilaatorist. Lahendus peab võimaldama kergesti välja vahetada keskkondi, kus CI konveiereid käivitatakse, näiteks virtualiseerimise abil.	
K2	Lahendus peab sisaldama integreerimisvõimalust ettevõttes hetkel kasutusel oleva koodivaramuplatvormiga. Integreerimisvõimaluste alla võivad kuuluda näiteks järgnevad võimalused: • Erinevate CI protsesside automaatne käivitamine kindlate sündmuste tagajärjel (näiteks projekti ehitusprotsessi käivitamine koodimuudatuste lisamise või tõmbetaotluse tekitamise tagajärjel). • Konveierite seadistamine, käivitamine ja nende tulemuste vaatamine otse läbi koodivaramuplatvormi kasutajaliidese.	
K3	Lahendus peab olema kuluefektiivne – lahenduse kasutamine ei tohi olla majanduslikult kulukam kui on eeldatav lahenduse kasutamisest saadav tulu.	
K4	Lahendus peab tulevikus võimaldama HIL-testimise kasutuselevõtmist.	
K5	Lahendus peab olema lihtsasti kasutatav ja hästi dokumenteeritud.	

Tähtsus	**Kirjeldus**
Must have	Kriteerium on kriitilise tähtsusega. Kui kasvõi üks *Must have* nõue on täitmata, ei ole antud lahendus sobiv.
Should have	Kriteerium lisab olulist väärtust, kuid pole kriitilise tähtsusega.
Could have	Kriteerium on vähetähtis (n-ö *nice to have*).
Won’t have (this time)	Kriteerium ei ole praeguses kontekstis oluline.

Lisa 3 – CI lahenduste kaalutud hinnangute arvutamine

Lahendustele kaalutud hinnangute andmine											
		Jenkins		**TeamCity**		**BuildBot**		**GitLab CI/CD**		**Azure Pipelines**	
Kriteerium	Kriteeriumi kaal	Hinnang	Kaal*hinnang	Hinnang	Kaal*hinnang	Hinnang	Kaal*hinnang	Hinnang	Kaal*hinnang	Hinnang	Kaal*hinnang
K1	**4**	5	5*4=20	5	5*4=20	3	3*4=12	5	5*4=20	5	5*4=20
K2	**4**	3	3*4=12	3	3*4=12	3	3*4=12	3	3*4=12	5	5*4=20
K3	**3,17**	4	4*3,17=12,68	0	0*3,17=0	4	4*3,17=12,68	1	1*3,17=3,17	4	4*3,17=12,68
K4	**3**	3	3*3=9	3	3*3=9	3	3*3=9	3	3*3=9	3	3*3=9
K5	**2,62**	5	5*2,62=13,1	3	3*2,62=7,86	3	3*2,62=7,86	4	4*2,62=10,48	3	3*2,62=7,86
Kokku			**66,78**		**48,86**		**53,54**		**54,65**		**69,54**

Lisa 4 – Dockeri süsteemipildi loomise skript

```
FROM ubuntu:20.04

# Make apt noninteractive and set a timezone
ENV DEBIAN_FRONTEND=noninteractive
ENV TZ=Europe/Tallinn
RUN ln -snf /usr/share/zoneinfo/$TZ /etc/localtime && echo $TZ > /etc/timezone

# Install dependencies
RUN apt -y update && apt install -y \
  build-essential \
  git \
  autoconf \
  libtool \
  pkg-config \
  gnupg \
  apt-transport-https \
  ca-certificates \
  software-properties-common \
  wget \
  python \
  python3 \
  clang-format \
  clang-tidy \
  cmake \
  libgmock-dev \
  libgtest-dev

# Get run-clang-format
WORKDIR /usr/local/bin
RUN wget \
  raw.githubusercontent.com/Sarcasm/run-clang-format/master/run-clang-format.py \
  && chmod 755 /usr/local/bin/run-clang-format.py
WORKDIR /

# Kitware repo for latest cmake
RUN wget -O - https://apt.kitware.com/keys/kitware-archive-latest.asc
    2>/dev/null | apt-key add -
RUN apt -y update && apt -y install cmake

# Add compilers
ADD gcc-linaro-7.4.1-2019.02-x86_64_aarch64-linux-gnu.tar.xz /opt
ADD armv5-eabi--glibc--stable-2018.11-1.tar.bz2 /opt
ADD gcc-arm-10.2-2020.11-x86_64-aarch64-none-linux-gnu.tar.xz /opt
ADD gcc-arm-none-eabi-9-2019-q4-major-x86_64-linux.tar.bz2 /opt
```

Lisa 5 – Azure Pipelines ehituskonveieri skript

```
trigger:
- master

pool:
  vmImage: 'ubuntu-latest'

jobs:
- job: Check
  container: rsoodla/c_cxx_base:latest
  steps:
   - checkout: self
     submodules: true
   - script: run-clang-format.py -r src
     displayName: Check code formatting

- job: Build
  container: rsoodla/c_cxx_base:latest
  steps:
   - checkout: self
     submodules: true

   - task: CMake@1
     inputs:
       cmakeArgs: '..'
     displayName: Generate build system

   - task: CMake@1
     inputs:
       cmakeArgs: '--build . --target install'
     displayName: Build sources

   - task: CMake@1
     inputs:
      cmakeArgs: '--build . --target test'
     displayName: Run tests
     continueOnError: true

   - task: PublishTestResults@2
     inputs:
       testResultsFormat: 'JUnit'
       testResultsFiles: '**/test_detail.xml'

   - task: PublishPipelineArtifact@1
     inputs:
       targetPath: '$(Build.SourcesDirectory)/build/dist'
       publishLocation: 'pipeline'
```