TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond
Arvutiteaduse instituut
Võrgutarkvara õppetool

Programmi koosteprotsessi automatiseerimine QuickBuildi abil

Bakalaureusetöö

Üliõpilane: Taavi Tali
Üliõpilaskood: 083871IAPB
Juhendaja: Jaagup Irve

Tallinn
2015

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.

_____ _____

(*kuupäev*) (*allkiri*)

Annotatsioon

Töö eesmärk on automatiseerida uue toote versiooni komplekteerimis- ja loomisprotsessi pideva integratsiooni serveris QuickBuild firma Videobet näitel. Selle tulemusel asendatakse senine manuaalne tegevus kiirema, veakindlama ja järelevalvet mittenõudva töövooga.

Töö käigus tehakse kindlaks komponendid, et koostada toode, seejärel luuakse ühene viis nende osade valimiseks ja teostatakse sammud, mis valiku realiseeriksid, kasutades QuickBuild'i poolt pakutavaid võimalusi. Viimaks uuritakse kasutuses olevate versioonimärkmete struktuuri ning kohandatakse seda kasutamiseks mallina.

Töö tulemusena valmib QuickBuild'i konfiguratsioon koos muutujatega ja sisemise töövooga, mille kaudu ehitatakse nõutav toode ja genereeritakse toodet kirjeldavad versioonimärkmed.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 52 leheküljel, 7 peatükki, 4 joonist, 2 tabelit, 3 koodinäidet.

Abstract

The goal of this thesis is to automate the process of creating and packaging a new product version in the continuous integration server Quickbuild used in a company named Videobet. This should result in current manual activity being replaced with a faster, less error-proof and autonomous workflow.

The work required determining the components needed for assembling the product, followed by designing a singular method for choosing between them and implementing the specified actions, all using options made available through QuickBuild. Lastly, it was necessary to analyze and customize the structure of release notes currently in use so it could be used as a template.

This work results in creating a QuickBuild configuration with accompanying variables and internal workflow, which are used to build the requested product and generate the release notes.

The thesis is in Estonian and contains 52 pages of text, 7 chapters, 4 figures, 2 tables, 3 code examples.

Lühendite ja mõistete sõnastik

Paindlik metodoloogia	***Agile methodology*** 12 printsiibile põhinev iteratiivne tarkvaraarenduse metodoloogia [1].
Pidev integratsioon	***Continuous Integration*** Tarkvaraarenduse praktika, mille kohaselt peale iga muudatuse (ingl. k. „commit“) versioonihaldussüsteemi tuleks kood automaatselt kompileerida ja testida.
Scrum	***Scrum*** Agiilne e. paindlik tarkvara arendamise meetod, milles töö jaotatakse kahe kuni neljanädalasteks tsükliteks e. sprintideks. Tsükli lõpuks tarnitakse kliendile toote inkrement, milles on lahendatud enne sprindi algust planeeritud kasutuslood (ingl. k. „user story“) [2].
JVM	***Java virtual machine*** Paljudele platvormidele porditud Java programmeerimiskeele kompileeritud baitkoodi interpretaator.
Groovy	***Groovy*** Objekt-orienteeritud skriptimiskeel, mida interpreteeritakse JVM-is või kompileeritakse Java baitkoodi (võimalik on kasutada Java keele teeke). Keel on dünaamiline ja imperatiivne, kuid toetab ka funktsionaalse programmeerimise vahendeid.
Maven	***Maven*** Deklaratiivne tarkvara ehitamise automatiseerimise rakendus. Projekti identifitseerib nn koordinaat kujul *groupId:artifactId:packaging:classifier:version*.
JSON	***JSON*** Andmevahetusformaat.

POM	***Project Object Model*** Maveni projekti mudel, mis kirjeldab kuidas projekti tuleb ehitada, reeglina *pom.xml*-is.
Nexus	***Nexus*** Failihoidla haldamise rakendus.
SNAPSHOT	***SNAPSHOT*** Termin, mida kasutatakse arendamisjärgus toote versioonis, nt *1.0-SNAPSHOT*.
Terminal	***Terminal*** Mänguautomaat kasiinotööstuses.
Kassiir	***Cashier*** Automaat, mida kasutatakse sellega ühendatud terminalide haldamiseks.
QA	***Quality Assurance*** Kavandatud süstemaatilised toimingud, mis on vajalikud, et tagada komponendi või süsteemi vastavust kehtestatud tehnilistele nõuetele.
LDAP	***Lightweight Directory Access Protocol*** Komplekt protokolle, mis võimaldavad ligipääsu infokataloogidele peaaegu igal rakendusel ja arvutil [3].
SVN	***Subversion*** Tsentraalne versioonikontrollisüsteem.

Jooniste nimekiri

Tabelite nimekiri

Sisukord

1. Sissejuhatus

1.1 Taust ja probleem

Viimastel aastatel on pidevalt kasvanud agiilse tarkvaraarenduse populaarsus. Üldjuhul on ettevõtted kasutusele võtnud klassikalise koskmudeli asemel mõne paindmeetodi raamistiku – Scrum, *Lean* jt [4]. Vastupidiselt koskmudelile, on agiilne arendusprotsess järk-järguline ja seega tarnitakse kliendile toote uus versioon tihedamini, nt Scrum'i praktiseerides keskmiselt iga 2-4 nädala tagant. Koguka produkti korral avaldab selline paradigma muutus tavapärasest rohkem survet nii toote ehitamisele (ingl. k. „building") kui ka kättetoimetamisele (ingl. k. „delivery").

Kasiinoturg on üldjuhul seadusandluse kaudu reguleeritud. Erinevate regulatsioonide tõttu võib ühe ja sama ettevõtte toode olla riigiti teistsugune. Mida rohkem turgudel firma tegutseb, seda kompleksemaks muutub tema poolt pakutav produkt. Tarkvara maailmas teisendub see enamasti rohkemate paralleelselt arendatavate koodiharude olemasolule. Uuele turule sisenedes võib uue seadistuse lisamisega töö piirduda, aga kui nõutav on toote põhifunktsionaalsuse muudatus, siis paratamatult peab koodihoidlas toimuma hargnemine (ingl. k. *branch out*).

Pideva integratsiooni keskkonnas eksisteerivad peale regulaarsete hetktõmmiste (SNAPSHOT-versioon), mis käivitatakse peaaegu iga arendaja koodihoidla muudatuse peale, ka avaldamisvalmis uuendused (ingl. k. „release"). *Release* versiooniga toodet või toote osa võib pidada valmis kliendile saatmiseks. Selliste versioonide loomist ei käivitata automaatselt, vaid vastavalt arendaja või vastutava isiku käsu peale. Reeglina lõpetab uuenduse loomise protsessi versioonimärkmed.

Võttes arvesse toote kasvavat mahukust, keerukust ja üha suurenevat erinevate aktiivsete koodiharude arvu, kulub üha enam aega ja inimressursi sellisele töövoole. Modulaarse e. tükeldatava toote puhul võib ette tulla ka juhuseid, mil kombineeritakse mitmest eri koodiharust ehitatud komponente, mis komplitseerib olukorda veelgi. Mida keerukam toode, seda tõenäolisem on, et inimese töösse tuleb sisse eksimusi.

Vähendades inimosa protsessis, siis võib eeldada, et tagajärjena selliste vigade esinemine väheneb [5]. Edukalt automatiseeritud protsess aitab töötaja aega säästa, kuna see osa tööst tehakse ära masina poolt. Reeglina teeb masin sama tegevust ka väiksema ajakuluga. Säästetud aega on võimalik kasutada muude, kas jooksvate ülesannete täitmiseks või tegeleda süsteemi edasiarendusega.

Autor töötab ettevõttes OÜ VideoB (pikemalt Videobet, edaspidi VB) *Build and Integration Team* (edaspidi BIT) koosseisus, kus on taoline probleem sujuvalt üles kerkinud ja seega tekkis vajadus lahenduse järgi.

VB on Playtech'i tütarettevõte, mis spetsialiseerub kasiino mänguautomaatide tarkvara (platvorm, mängud) arendamisele [6] ning mille toodet kasutavad kliendid – kasiinode operaatorid - asuvad Euroopas, Aasias, Põhja- ja Lõuna-Ameerikas.

Lahendus on arendatud autori poolt Tallinnas, VB kontoris, alates 2014. aasta lõpust põhitöö kõrvalt. Kasutatav versioon valmis 2015. aasta mai alguseks.

1.2 Ülesande püstitus

Töö eesmärk on kasutatavas pideva integratsiooni serveris – QuickBuild – üles seada nn *one-click-release* konfiguratsioon, mis võimaldaks projektijuhi poolset toote nõuet kergelt täita. Lõplik lahendus peaks olema:

- Senisest (manuaalsest) protsessist väiksema ajakuluga
- N-ö *fire-and-forget*, s.t pärast andmete sisestamist ja protsessi käivitamisest kuni lõpliku väljundini (versioonimärkmed) on tegevus täielikult automatiseeritud
- Veakindel – esineb vähem eksimusi stiilis, et mõni nõutud komponent jääb ehitamata või versioonimärkmed sisaldavad ebatäpset informatsiooni.

Samuti peaks tööl olema kaudne kasu ka teistele ettevõtte osakondadele, kuna BIT on üks lüli terves tööahelas, seega tiimi töö mõjutab otseselt ka teisi. Mida lühem on BIT-il kuluv aeg uuenduste välja saatmiseks, seda kiiremini liigub toode oma elutsüklis edasi jõudes seetõttu varem ka kliendi kätte.

1.3 Metoodika

Käesolevas töös antakse esmalt ülevaade tehnoloogiatest, millega pidi autor lahenduse väljatöötamiseks kokku puutuma. Seejärel kirjeldatakse kuidas VB toode ja selle alamkomponendid on QuickBuild'iga seotud.

Saanud lühikese ülevaate süsteemist, alustatakse eesmärgis püstitatud konfiguratsiooni üles seadma, jooksvalt välja tuues QuickBuild'i poolt pakutavaid võimalusi ja eripärasid.

Kuna lahenduse välja töötamisel kulus rohkem aega muutujate ja astmete loomiseks ja seadistamiseks, siis neid käsitleb autor põhjalikumalt.

Muutujate seadistamise kaudu tekitatakse konfiguratsiooni käivitamise alguses kuvatavad valikuväljad, mille kaudu on kasutajal võimalus täpsustada ehitatavad toote osad. Töösse on kaasatud autori poolt kirjutatud koodi näiteid, ekraanitõmmis muutujate täpsustamise kuvast, ülevaatlikud tabelid loodud muutujate ja astmetega.

Kasutatakse peamiselt kahte tüüpi astmeid kahe erineva otstarbe jaoks - ühte kasutaja poolt valitud komponentide loomiseks ja teist ehitatud komponentide kohta informatsiooni muutujatesse salvestamiseks.

Töö lõpus käsitletakse versioonimärkmete malli põhjal loomist vastavalt konfiguratsiooni muutujate täpsustamise kuvast valitule.

2. Kasutatavad tehnoloogiad

2.1 QuickBuild

QuickBuild (edaspidi QB) on üks paljudest tänapäeval levivatest pidevat integratsiooni praktiseerimist võimaldavatest tarkvaralahendustest. Tuntumad samalaadsed tooted on Jenkins [7], TeamCity [8], Bamboo [9] jpt. QB on sarnaselt Jenkins'iga mitme platvormi toega eraldiseisev serveritüüpi rakendus, nn pideva integratsiooni server (ingl. k. „continuous integration server") [10].

QB põhiliseks osaks on konfiguratsioonid. Igat sellist indentifitseerib tee, nt *vb/new/component/release_jira/2.00*. Konfiguratsioonid moodustavad hierarhilise süsteemi, s.t igal osal on seaded, nt hoidlate (ingl. k. „repositories") sätted ja muutujad, mida saab hierarhias madalamal astmel asuja kasutada (või üle kirjutada) [11]. Näiteks *vb/new/component/Backend* ja *vb/new/component/Frontend* teega konfiguratsioonid pärivad mõlemad oma vaikeseaded *vb/new/component* nn emakonfiguratsioonist. Kusjuures enamik muudetavatest väljades saab kasutada kolme erinevat skriptikeelt: MVEL, Groovy, OGNL. Seeläbi on võimalik konfiguratsioone igat pidi kohanda ja dünaamiliseks muuta.

Selle süsteemi mõistes on konfiguratsioon eeskiri, mis kirjeldab, kuidas *build agent* peaks *build*'i genereerima. Taolise eeskirja e. töövoo osa on *step,* mida töövoogu lisades järjestikku teostatakse. QB-ga on kaasas hulk taolisi sisseehitatud „astmeid", millest põhilised on koodihoidlast lähtekoodi allalaadimine, *build*'i automatiseerimistarkvara (Ant, Maven) käsu käivitamine või manusskripti (MVEL, Groovy, OGNL) jooksutamine. Tavajuhul on *build*'i sisuks tarkvara projekti kompileerimine, testide jooksutamine, seejärel väljundi pakendamine ja paki paigutamine failihoidlasse (Nexus) ning viimaks teavituse saatmine arendajatele. Loodavas konfiguratsioonis on põhiliselt kasutatud teise konfiguratsiooni käivitamise ja (Groovy) skriptide jooksutamiseks ette nähtud astmeid.

QB on suletud lähtekood ja täisversiooni kasutamiseks tuleb soetada litsents, kuid on võimalik proovida ka tasuta varianti, mille kasutatavate konfiguratsioonide arv on piiratud.

Töös on kasutatud QB 4.0 versiooni.

2.2 JIRA

Atlassian JIRA on veebipõhine projektijuhtimise ja probleemihalduse (ingl. k. „issue tracking“) tarkvara. Rakendus võimaldab tööd organiseerida ja saada jooksvalt ülevaade projektist kogu tema eksisteerimise vältel. JIRA keskmes on probleemid (ingl. k. „issue“), mida on sõltuvalt eesmärgist mitut eri tüüpi [12]. VB-s on peamisteks: *defect, delivery, epic* ja *build*. Viimane on reeglina suunatud BIT-le ning mida kasutatakse suvalise komponendi (või kogu toote) ehitamise soovi edastamiseks. Igal *issue*'l on väljad, mis kirjeldavad täpselt probleemi olemust. Seisundiväli näitab, et mis elutsükli osas probleem parasjagu asub. VB-s on kasutusel kokku kaheksa (8) erinevat seisundit:

1) *Open* – probleem on avatud ja määratud isikul on võimalus selle kallal tööle asuda
2) *In Progress* – määratud isik tegeleb aktiivselt probleemiga
3) *On Hold* – töö on mingil põhjusel peatatud
4) *Resolved* – probleem on saanud lahenduse, nt arendaja on koodimuudatusega vea parandanud
5) *Reopened* – probleem oli *Resolved* või *Closed* staatuses, aga nüüd on taasavatud
6) *Unverified* – probleem on küll lahenduse saanud, aga lahendust pole keegi (QA osakonnast) kontrollinud
7) *Verified* – lahendus on kontrollitud
8) *Closed* – töö probleemi kallal on lõpetatud.

JIRA Agile laienduse kaudu saab praktiseerida paindliku metoodika erinevaid raamistikke – Scrum, Kanban [13]. Laiendit saab kasutada tiimidele scrum-tahvlite, kus igas veerus on vastava seisundiga probleem, loomiseks. Tahvel kuvab jooksvalt ülevaadet tiimi tööst ning võimaldab hõlpsalt probleemi seisundit muuta [14]. Ka BIT kasutab sellist, sinna tekivad projektijuhtide poolt loodud *build*-tüüpi probleemid (*build request*), nõnda andes märku, et toote ehitamisega võib alustada. Töö alustamisel liigutab BIT-i liige probleemi *Open* veerust *In Progress* veergu.

Kusjuures peab sellisesse probleemi olema lingitud teisi (alam)probleeme, mis näitavad, et nende lahendamise tõttu (e. asuvad 4., 6., 7. või 8. seisundis) soovitakse toote uut versiooni.

2.3 Velocity

Velocity on Apache vaba tarkvara projekt, mis hõlmab endas Java programmeerimiskeeles kirjutatud mallimootorit (ingl. k. „template engine“) nimega Velocity Engine [15], mida saab kasutada igasuguse tekstipõhise faili (XML, e-kiri, HTML jne) genereerimiseks. Velocity võimaldab viidata Java objektidele ja nende sisu vastavalt mallile kuvada, olles oluline lüli vaate ja programmi loogika vahel.

Mootor toetab mallides muutujaid, valiktingimusi ja tsükleid. Käsud algavad #-märgiga. Velocity Tools on sama Apache projekti alla kuuluv laiendus, mis võimaldab lisamisel kasutada *math, date, lists* jt teeke.

Töös kasutatav QB versioon kasutab sisemiselt Velocity 1.5-te.

2.4 VBPacker, VBPack ja VBPack Launcher

VBPacker on VB-siseselt arendatud Maven’i plugin, mida kasutades on võimalik luua sellise sisuga VBPack-faile, mida on võimalik kasutada terminali või kassiiri installimiseks.

VBPack (edaspidi ka pakk) on *vbpack*-faililaiendusega arhiiv, mille sisse on paigutatud plugina kasutust kirjeldav *pom.xml* ja Maven’i üldsätteid deklareeriv *settings.xml* [16] fail. Seega on pakk oma olemuselt Maven’i projekt.

Pom.xml failis on välja toodud paki poolt kasutatavate komponentide Maven’i koordinaadid [17], mille versioon on fikseeritud. Toote tuuma – miinimumkomplekti toote käivitamiseks - moodustavad *Core* XML märgendi alla kuuluvad koordinaadid.

VBPack Launcher on rakendus VBPack’ide käivitamiseks ning oskab arhiivi sisus olevat *pom.xml* VBPacker plugina töökirjeldust täita, s.t kuvab *pom.xml*’is deklareerituid valikuid (kliendi-, serveri- ja terminali/kassiirikonfiguratsioon jt) ja annab võimaluse ka vastava kirjelduse järgi *content*’i – mängude, teemade ja *lobby*’de - lisamiseks. Segaduste vältimiseks *Core* komponente Launcher’i kaudu muuta ei saa.

Launcher'i töö päädib sellega, et kasutajalt küsitakse kuhu ja kuidas (valitavad meetodid täpsustab samuti *pom.xml*) kõik kasutaja poolt valitud komponendid ja toote tuumik paigutatakse.

3. Toote ülevaade

VB arendab serveripõhist kasiino mänguautomaatides jooksvat terviklahendust. Toote võib üldjoontes jagada neljaks põhiosaks: *frontend, backend, configuration* ja *content*.

Kõiki harude alamosi ehitatakse QB kaudu ning nende komponentide juurkonfiguratsioonid asuvad konfiguratsiooniteedel kujul *vb/new/component/{mainComponent}/{branch}*, kus *{mainComponent}* on üks neljast põhikomponendist ja *{branch}* tema versioon.

Neljast põhiosast moodustatakse alljärgnevad eraldiseisvad VB tooteplatvormi kuuluvad tükid:

- Terminal
- Kassiir
- Server – keskne komponent, mille ühenduseta terminal ja kassiir tegutseda ei saa ning vastutab ühendatud terminalide ja kassiiride äriloogika seadistamise eest
- Proxy – proksid teatud serveri funktsionaalsuse matkimiseks
- Office – kassiiri funktsionaalsust pakkuv veebiliides
- Mängud – erinevad SD ja HD platvormi kasiinomängud
- Teemad (*themes*) – kliendi spetsiifilised koos nende paigutust ja kasutust täpsustavate failide kogumikud
- Lobby'd – graafika, mida kasutatakse terminalis olevate mängude kuvamiseks mängijale
- Konfiguratsioon – klientide, terminali- ja kassiiritüüpide ning serverite seadistusfailid.

Iga loetelus välja toodud element on QB-s eraldi ehitatav. Vastava konfiguratsiooni töö lõpus luuakse

Terminal ja kassiir koosnevad peaasjalikult eessüsteemi osadest (*vbpack*), kuid nendega käib kaasas eraldi *backend* komponent serveriga suhtlemiseks. Mänguautomaadile paigutatakse

vastavalt kliendi soovile sisu (*content*) – mängud, teemad, *lobby*'d. Tavamängijal on juurdepääs vaid terminalile ja selles olevale sisule. Mängualas on klienditeenindajal lisavõimekus kassiiri olemasolu tõttu - see annab talle terminalidest ülevaate ja nende haldamise võimekuse. Operaatorile on tagatud kassiirilaadne funktsioon ka brauseriga ligipääsetava veebiliidese (Office) kaudu. Kuna turud on erinevad ja pole ka olemas kahte samasugust klienti, siis peab toode olema seadistatav. Üks operaator võib kasutada hoopis teistsugust tüüpi mänguautomaate kui teine. Kõik kliendid on ka eraldiseisvad – neile peab olema oma server, kuhu automaadid ühenduvad. Võttes neid ja ka mitte mainitud asjaolusid arvesse, siis peavad terminal, kassiir ning samuti server olema eraldi konfigureeritavad.

4. Konfiguratsioon

Omades ülevaadet QB-s ehitatavatest toote komponentidest, asuti looma *one-click-release* konfiguratsiooni, millega saaks töö ülesandes püstitatud eesmärke täita. Konfiguratsiooni seadistamisel langes põhirõhk uute muutujate ja astmete loomisele.

4.1 Konfiguratsiooni muutujad

Igal konfiguratsioonil on komplekt muutujaid, mis võimaldavad jooksvalt kohandada käivitatavat *build*'i kasutades neid astmetes, hoidlate sätetes jne. Muutuja võib olla eelväärtustatud või seatud küsima väärtust *build*'i alustamisel. Küsitavat väärtust saab täpsustada muutuja *Prompt Setting* valikuvälja kaudu. Loodud konfiguratsioon kasutab kolme erinevat valikuvälja: tekstisisend (*text*), loendivalik (*selection*) ja mitme valiku väli (*multi-selection*).

Kasutaja võib otsustada ka valikuvälja midagi sisestada või valida. Pannes muutuja *Allow Empty* valikuvälja linnukese, tagab QB, et enne konfiguratsiooni käivitamist määratud valikuväli ei saa olla tühi.

4.1.1 Muutujate täpsustamise kuva

Konfiguratsiooni jooksutamise nuppu vajutades kuvatakse kasutajale lehekülg (vt Joonis 4), kus tuleb täpsustada ehitatavad komponendid. Kokku kuvatakse 18 erinevat andmevälja (vt Tabel 1). Erinevaid komponente on kokku üheksa, neist kaheksa on kuvatud *Components* lahtris (vt Tabel 1). Iga mitme valiku väljast valitud komponendiga käib kaasas koodiharu või alamkomponendi täpsustamine, s.t valituks osutumisel täitub ka vastav rippmenüü sisu (konfiguratsiooni hierarhiast). Analoogne kasutusjuht on välja toodud ka QB dokumentatsioonis [18], kus on viidatud ka taolist muutujate sisu dünaamilisust esitlev demo. Selle töö teeb muutuja seadetesse *Choices* boksi manustatud Groovy skript (vt Joonis 3). Joonisel on näha, et *frontend* väärtuse sisaldumisel *componentsToBuild* lahtris, otsitakse QB-st kõik *vb/new/component/Frontend* teega konfiguratsioonide alamkonfiguratsioonide nimed.

Täitmiseks kohustuslikuks on määratud kõik väljad. Selle tingis asjaolu, et kasutaja vaataks läbi iga välja ja nõnda välistades, et mõni väli jääb ekslikult täitmata. Nn tühja valiku jaoks on võimalik valida *None* väärtus, mis tähendab, et vastav komponent on küll versiooni osa, aga

selle osa ehitamist ei alustata. Selline vajadus tekib kui eelnevalt on osa ehitatud ning soovitakse komponendi info kaasamist versioonimärkmetesse.

4.2 Konfiguratsiooni astmed

Loodud konfiguratsiooni töövoo koostamisel on kasutatud kuute eri tüüpi astmeid:

- *Trigger Other Builds* – astme seadete *Configuration* välja määratud QB konfiguratsiooni käivitamine
- *Execute Script* – (Groovy) skripti käivitamiseks
- *Checkout* – versioonihaldussüsteemist lähtekoodi või failide allalaadimine
- *Artifacts* – failide avaldamiseks, seejuures tehes need kättesaadavaks Velocity mallile
- *Sequential* – astme sisse paigutatud astmete järjestikku jooksutamiseks; vaates paigutuvad vertikaalselt
- *Parallel* – sama, mis eelmine, aga alamastmed käivitatakse paralleelselt; vaates paigutuvad horisontaalselt.

Ehkki iga astet saab jooksutada eraldi masinas, siis käesolevas konfiguratsioonis käivitatakse kõik serveri poolt, kuna loodud astmed on väikese ressursinõudlusega ja osa skriptidele kättesaadavaks tehtud rakendusliidesest tavalises masinas ei funktsioneeri. *Trigger Other Builds* käivitatakse konfiguratsioon selle seadetes ressursi järgi, milleks on üldjuhul määratud vastupidiselt serverile *build* masin.

Konfiguratsiooni töövoogu saab mõjutada nii programselt (astme *Execute Condition* kriteeriumid) kui ka käsitsi. Astmeid saab individuaalselt ümber tõsta või klikivajutusega kinni keerata või uuesti lubada.

Esimese taseme astmeid jooksutatakse järjestikku (*Sequential*) ja tingimusel, et eelnev aste on olnud edukas. Seega ühe astme ebaõnnestumisel järgnevate astmeteni ei jõuta ja *build* lõpetab ebaõnnestunult. Erandiks on *Parallel*-tüüpi *step*. Sel juhul tema alamastmed käivad üksteisest sõltumatuna, s.t kui üks alamaste lõpetab edutult, siis teised töötavad ikkagi edasi. Kui alamastmed töö lõpetavad, siis paralleelne aste ebaõnnestub, kuna on seadistatud nii, et kasvõi ühe lapsastme mitteedukal lõppemisel käitub vanem samamoodi. Nii lõpetab kogu *build* oma

töö varem ja saab järgneda vea otsimine. Sama koha pealt saab jätkata nii, et konfiguratsiooni valikuväljadesse valitakse küll samad komponendid, mida ehitada (*componentsToBuild*, *backendComponents*), mis eelnevalt, kuid *Steps* vaates keeratakse eelmises *build*'is õnnestunud komponentide käivitamise astmed kinni.

4.2.1 JIRA versiooni avaldamine

Iga toote, mis on küll kompleksne, väljalaskega käib kaasas kõiki eri komponente üheselt identifitseeriv versioon, mille malliks on {*majorVersion*}.{*minorVersion*}.{*customerVersion*}.{*buildVersion*}. Kõiki taolisi versioone hoitakse JIRA-s ning kui esmase versiooni, nt *2.05.000.000*, tekitamise eest hoolitsevad projektijuhid, siis iga järgnev inkrement, st *2.05.000.001*, tekitatakse QB poolt.

Selle tarbeks on olemas QB-sse sisseehitatud *step* nimega „*Release JIRA versioon"*. Astet kasutatakse selleks pühendatud konfiguratsioonipuus, millest hierarhias allpool on põhiversioonide - *majorVersion* ja *minorVersion* - nimedega alamkonfiguratsioonid, nt *2.00*. Nendest käivitatud *build*'ide versioonid on seatud kattuma eelnevalt mainitud versioonimalliga ning seega dubleerivad JIRA-s asuvaid versioone. *Step*'i *Extra Attributes* väljale on lisatud *build*'i versiooniga väärtustatud *fixVersion* atribuut. QB, kasutades sisemiselt JIRA rakendusliidest, avaldab vastava projektiversiooni ning märgib selle versiooni raames *Resolved/Closed* staatusesse jõudnud probleemide „Fix Version/s" väljad sama versiooniga.

Vastava JIRA versiooni avaldamise on loodud *trigger-jira-version-build* aste (vt. Tabel 1), mis kasutab muutuja *jiraVersion* sisu käivitatava konfiguratsiooni tee otsustamisel ning väärtustab *jiraLongVersion* muutuja käesoleva alampeatüki alguses kirjeldatud versiooniga.

4.2.2 Valitud konfiguratsioonide käivitamine

Kõigist loodud astmetest (vt. Tabel 1) on 1. ja 3.-27. järjekorranumbriga eesmärgiga käivitada kasutaja poolt valitud konfiguratsioonid, s.t alustada nende ehitamist. Kuna osa komponente pole omavahel seotud, siis saab neid ehitada paralleelselt (*Parallel* astmetüüp). Nõnda väheneb kogu protsessile kuluv aeg.

Tootes on ka osi, mis on omavahel rangelt seotud, nt *frontend*'i binaarfailid ja GAM-id, seejuures tuleb oodata esimese töö alamvoo lõppu kui saab asuda teise kallale. Sama kehtib ka osade protsesside, nt *backend*'i komponentide juurutamise puhul – enne tuleb uus versioon

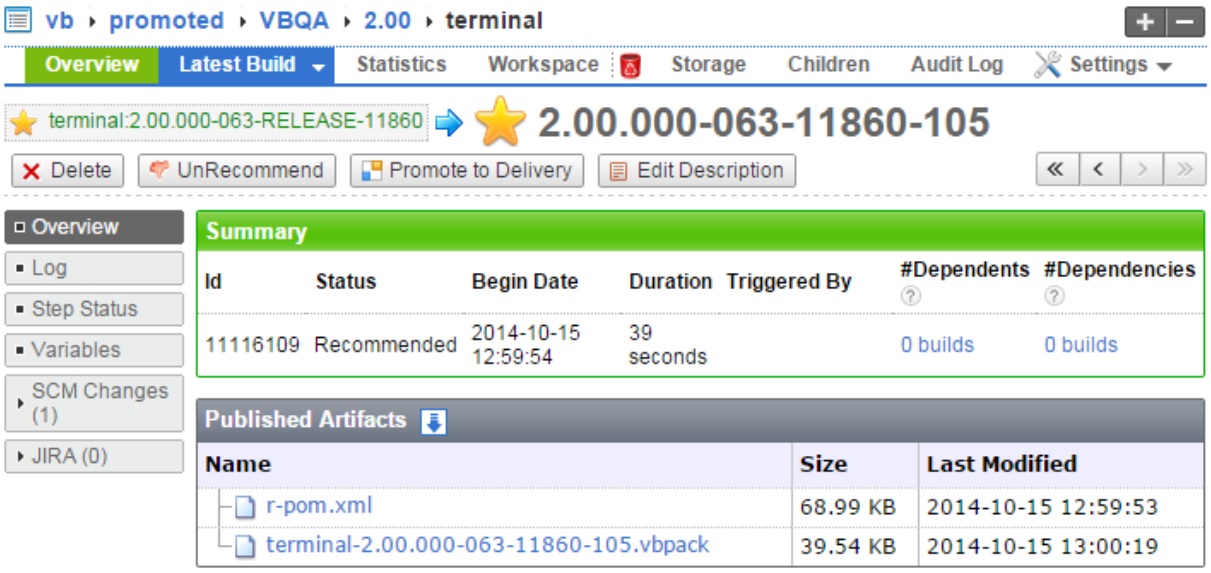
ehitada ja alles seejärel saab seda versiooni testserverisse paigutada. Sama mõtet tuleb ka järgida *vbpack*'i ülendamisel – enne *trigger-{cashier|terminal}-vbpack-promotion* (vt Tabel 2) astet tuleb uus *vbpack* ehitada. Ehkki tegemist on veatekke ohukohaga, eksisteerib ka kasutusjuhte, mille puhul saab eripära ära kasutada (eelmise näite puhul vana *vbpack*'i lihtsalt uuesti ülendada) kui kinni keerata vastav aste konfiguratsiooni töövoos.

4.2.3 VBPack'i ülendamine

Juhul kui terminali ja/või kassiiri *release* pakk valmib edukalt, siis on vajalik muuta *vbpack*'e nii, et need oleks QA osakonna jaoks kasutajasõbralikumad. Seega tuleks eemaldada valikud, mis pole QA-le eriomased, nt BIT-i tööriistad või konfiguratsioonid, mis pole suunatud QA test serverite vastu.

Ülesande lahendamiseks on kasutatud QuickBuild'i *build promotion* [19] funktsionaalsust. Konfiguratsiooni *Promotions* seadete alla lisatakse uus definitsioon (ülendamised on päritavad, seega on ka neid mõistlik luua emakonfiguratsioonis). Uue lisamisel tuleb seadetes täpsustada, millisel juhul on võimalik käesolevat *build*'i ülendada ja mis on sihtkonfiguratsiooniks, s.t millist konfiguratsiooni tuleks startida. Salvestamise tagajärjel tekib tingimusi täitvate *build*'ide ülevaate lehele lisanupp *promotion*'i nimega või rippmenüü (rohkem kui ühe valiku korral). Nupuvajutusega käivitatud ülendamine kujutab endast tavalise *build*'i alustamist sihtkonfiguratsioonist. Loodava *build*'i eripära on see, et koodihoidlate versioon (ingl. k. „revision") samastatakse ülendamist alustatud *build'*i omadega.

Vbpack'i ülendamisel tekitatakse sihtkonfiguratsiooni allalaetavate failide valiku alla uus QA-le suunatud variant sellest (vt Joonis 1). Kusjuures joonisel nähtava *vbpack*'i allalaadimislingi saamiseks Groovy'ga tuleb kasutada käsku *_build.getDownloadUrl('artifacts/terminal-2.00.000-063-11860-105.vbpack', true)*, kus *_build* on selle konkreetse *build* objekt (com.pmease.quickbuild.model.Build klass).



Joonis 1. VBQA konfiguratsiooni ülendatud terminali *build*

Programselt on ülendamine lahendatud terminalile ja kassiirile sisu poolest sama *Execute Script step*'iga (vt Tabel 2), milleni jõutakse pärast vastava valikurohkema BIT *vbpack*'i ehitamist. Nende astmete nimed on vastavalt *trigger-terminal-vbpack-promotion* ja *trigger-cashier-vbpack-promotion* (vt Tabel 2), kusjuures nime sisu kasutatakse skriptis *step.name.split('-')[1]* (tagastab esimese puhul *terminal* ja teise korral *cashier*) kaudu *build*'i *step.name.split('-')[1] + VbpackLink* nimega muutuja väärtustamiseks.

4.2.4 Põhikomponentide andmed

Manuaalselt versioonimärkmete koostamisel kulus enamasti kõige rohkem aega ülendatud kassiiri ja terminali pakis kasutatavate põhikomponentide kohta info kogumiseks. Selleks tuli avada pakk VBPack Launcheriga ja kopeerida *Core* info (kasutatud SVN muudatuse versioon puudub) kirja, seejärel otsida üles QB konfiguratsioonihierarhiast komponendile vastav *build* ning leida sellest logist muudatuse versioon.

Protsessi kiirendamiseks loodi *get-revisions* aste (vt Tabel 2), mis automatiseerib otsingut ning mille tulemusena salvestatakse vastav info *terminalComponents* ja *cashierComponents* muutujasse.

Loodud skript (vt. Koodinäide 2) laeb eelnevas peatükis loodud (4.2.3) paki(d) alla, loeb arhiivist *pom.xml* sisu ning sõelub neist *Core* XML märgendi alla kuuluvad Maven'i koordinaadid. Kuna VB-s seab iga (v.a ülendatud) *build* oma nimeks versiooni (täpsemalt RELEASE lõpuga), mis selle käitamise aegu loodi, siis saab nimed koordinaatide *version*

väljadega vastavusse viia. Seega jõuti järeldusele, et omavahel loogiliselt ühenduses oleva *build*'i muudatuse versiooni saab kasutada vastava koordinaadi kirjeldamiseks. Tulemus salvestatakse *terminalComponents* ja *cashierComponents* muutujatesse.

Arvestades QB konfiguratsioonide koguarvu (töö tegemise hetkel mitu tuhat) ja iga konfiguratsiooni keskmiselt paarikümmet salvestatud *build*'i, ei ole otstarbekas alustada taolist otsingut hierarhia kõige kõrgemast astmest. Sellise, nn jõumeetodi, asemel kasutatakse samas skriptis *Map*-tüüpi andmestruktuuri, mis on järgnevas koodilõigus nimega *confMap*.

```
…
// either (configurationName) or (componentArtifactId:configurationName), where
configurationName == configuration.getName()
def frontendComponents = ['engine', … ]
def backendComponents = ['com.videobet.cashier.product:cashier', … ]
…
// these paths will contain 'None' if the corresponding variable has not been
declared, which will be replaced when searching for the artifact version's branch
def frontendComponentsKey = 'vb/new/component/Frontend/' +
(vars.getValue('frontendBranch') ?: 'None') + '/binaries/'
def backendComponentsKey = 'vb/new/component/Backend/' +
(vars.getValue('backendBranch')  ?: 'None') + '/'
…
confMap = [
…
    (frontendComponentsKey) : frontendComponents,
    (backendComponentsKey) : backendComponents,
…
]
…
```

**Key* (võti) lõpuga muutujad – konfiguratsiooni teed – on seotud **Components* (väärtus) lõpuga muutujatega – koordinaatide *artifactId*'id. Väärtuste hulgast otsitakse välja vastav võti, mida kasutatakse *build*'i otsimise otseteena.

4.2.5 Versioonihalduse muudatused

Kui valitud komponentide ehitamine on jõudnud lõpule, siis jõutakse *get-svn-logs* astmeni (vt Tabel 2). Selles astmes hangitakse Groovy skriptiga SVN versioonihalduse muudatused igale kasutades käsurida kujul *svn log svnPath fromRev:toRev -v --xml --username user --password pw* [20], kus:

- *svnPath* – komponendi ehitamisel kasutatud koodihoidla URL-tee
- *fromRev* – ehitatud komponendile eelnenud versiooni (mitte-SNAPSHOT) *build*'is kasutatud koodihoidla versioon

- *toRev* – versioon, mis komponendi ehitamisel kasutati
- *user, pw* – automaatkasutaja andmed.

Kusjuures, kui on ehitatud rohkem kui üks alamosa ühe komponendi - *backend, frontend, configuration* – raames, siis väärtustatakse *fromRev* alamosade poolt väikseima ja *toRev* suurima muudatuse versiooniga vastava komponendi koodihoidlas. Sel viisil on muudatuste ulatus võimalikult lai ning ebatõenäolisem, et versioonimärkmetes on vale informatsioon.

Tagastatavat XML-i sõelutakse välja muudatuse versioon, autor, kuupäev, teade ja muudetud failide rajanimed ning salvestatakse vastava komponendi jaoks mõeldud muutujasse – *backendSvnChanges, frontendSvnChanges* või *confSvnChanges*.

4.2.6 JIRA lingitud probleemid

Get-jira-linked-issues aste (vt Tabel 2) eesmärk on hankida käivitamisel täpsustatud JIRA *build request*'i (*jiraIssue* muutuja) kõik alamprobleemid. Selleks on kasutatud JIRA REST-rakendusliidese otsinguressursis JQL-päringut [21]. Tagastatav tekst on JSON-formaadis, millega ümber käimiseks on lisatud QB-sse Json-lib teek [22]. Teegi poolt pakutavad klassid ja funktsioonid lihtsustavad oluliselt vastusest JIRA probleemi andmeväljade sõelumist. Eraldatud andmed salvestatakse *linkedJiraIssues* muutujasse, aga kuna iga QB *build*'i muutuja saab salvestada ainult stringina, mitte näiteks andmeobjektina ega ka objekti serialiseerida-deserialiseerida (Velocity'l tugi puudub), siis tuleb andmeväljad kodeerida valitud eraldajaga. Lisaks ei leidnud autor võimalust dünaamiliselt muutujate tekitamiseks, seetõttu tuleb kõik lingitud probleemid paigutada ühe muutuja sisse, s.t eraldada omakorda järgmise eraldajaga (nt *n*).

4.2.7 *Backend*-komponendid

Kui konfiguratsiooni käivitades on ehitamiseks valitud ka *backend* komponente, mis ei kuulu terminali ega kassiiri *Core* koordinaatide sekka, siis tuleb ka nende allalaadimislink salvestada, kuna versioonimärkmetes peab kuidagi need failid kättesaadavaks tegema.

Ülesande täitmiseks hangitakse ehitatud *backend* konfiguratsioonist *build* objekti (com.pmease.quickbuild.model.Build klass) *renderLogAsText()* meetodi abil tagastatud logi teksti peal kasutatakse regulaaravaldist *(?<=Uploading:\s).*\.zip*. Sellist avaldist saab kasutada, kuna iga Maveni kaudu üleslaetav fail prinditakse sellise algusega väljundisse ja kõik otsitavate komponendite logid sisaldavad ainult ühte sellist rida.

Leitud link salvestatakse *{var}Link* muutujasse, kus *{var}* on otsitav tagarakenduse komponent.

5. Versioonimärkmed

Iga toote versiooniga käib kaasas versioonimärkmed. Selle eesmärk on kõikidele osapooltele anda täpne ja kompleksne ülevaade loodud produktist. Märkmetesse kogutakse kokku oluline, olles seejuures konkreetne ja täpne. Samas kui avaldatavas toote versioonis on midagi eripärast või võrreldes eelmise versiooniga suurem muudatus sisse viidud, siis tuleks vastavaid asjaolusid ka mainida. Kui tegemist on pika kirjeldusega, siis tuleb piirduda viitamisega kasutades hüperlinki.

VB-s pannakse versioonimärkmed kirja meilina ja saadetakse meililisti, kuhu kuulub QA osakond kui ka arendajad (*backend, frontend, content*), Support, Deployment ning projektijuhtide tiimid.

Firmasisene ametlik suhtluskeel on inglise keel ja seega on nõutud, et ka kirja sisu oleks samuti.

5.1 Struktuur

VB-s eksisteerivad erinevad versioonimärkmete kirjade struktuurid sõltuvalt sellest, mis toodet soovitakse avaldada. Näiteks mängude (sh platvormi väliste) märkmed koosnevad vähema arvuga infoväljadest ning on seega oluliselt lühemad kui platvormi omad. Kuna käesolev töö ei käsitle mänge ja muud *content*'i, vaid tervet platvormi, siis keskendutakse vaid viimase versioonimärkmete mallile. Ehkki nii uue ja kui ka vana platvormi versiooni jaoks eksisteerib eraldi kirja struktuur, kavatsen vaid käsitleda ainult uuemat, sest vanemat QB kaudu ei ehitata ja on seega töö skoobist väljas.

Toote modulaarsus dikteerib, et ka väljastatav kiri oleks samuti osadeks jaotatav. Eriti aktuaalne on see agiilses tööprotsessis, kus tuleb tihti ette olukordi, mil on vaja ehitada ainult teatud osa tervest tootest, nt *frontend* või *configuration*. Sel juhul infot muude komponentide – *backend, configuration* jne - kohta ei ole vaja esitada. Seeläbi on lahendatud ka probleem toote erilahendustega, mille puhul klient näiteks ei soovigi kassiiri või veebiliidese olemasolu – muutujate valiku lehel seda komponenti ei valita.

Postkasti sisu sirvides on oluline, eriti kui eelvaade pole sisse lülitatud, et kirjade teemad annaks rohkem infot kui ainult, et tegemist on versiooni väljalaskega. Selleks tuuakse teemas välja toote versioon, mis klapib JIRA-s avaldatuga, ning selle raames ehitatud komponendid. See on vormindatud nii: *<komponendid> build <JIRA versioon>*. *<Komponendid>* on komadega eraldatud, nt *Terminal, cashier*. Juhul, kui on ehitatud kõik toote osad, siis kirjutatakse komponentide nimistu asemel lihtsalt *Full*, nt *Full build 2.05.000.000*.

Kirja põhilise teabe leiab selle sisust. Analüüsides minevikus välja saadetud kirju, siis täheldatud sisu muutuvad osad on:

1) Toote versioon
2) Link JIRA nn *build request*'ile
3) Lingid VBPack'idele
4) 2. punktis lingitud VBPack'ide iga tuumikkomponendi Maven'i koordinaat [17], mille järele on lisatud komponendi ehitamiseks kasutatud muudatuse versiooni
5) Lingid tagarakenduse osadele (server, Office, proksid)
6) JIRA probleemid, mis käesoleva versiooniga on lahendatud
7) Uued koodimuudatused kasutatavatesse koodihoidlatesse

ja osad, mis muutuvad harva:

1) Lühike juhend VBPack'i ja VBPack Launcher'i kasutamiseks
2) Test serveri aadress, mille vastu toode on üles seatud
3) Viide *content*'i pakkimise tööriistale.

5.2 Loomisprotsess

Kui kõik *step*'id on lõpetatud ja *build* lõpule jõudnud, alustab QB konfiguratsiooni *Notifications* seadete all määratud meili saatmist. Eelmises peatükis (5.1) välja toodud muutuva osa info on talletatud konfiguratsiooni astmete käigus vastavate nimedega *build*'i muutujatesse:

1) *jiraLongVersion* (vt 4.2.1)

2) *jiraBuildIssueLink* (vt Koodinäide 3)

3) *cashierVbpackLink, terminalVbpackLink* (vt 4.2.3)

4) *cashierComponents, terminalComponents* (vt 4.2.4)

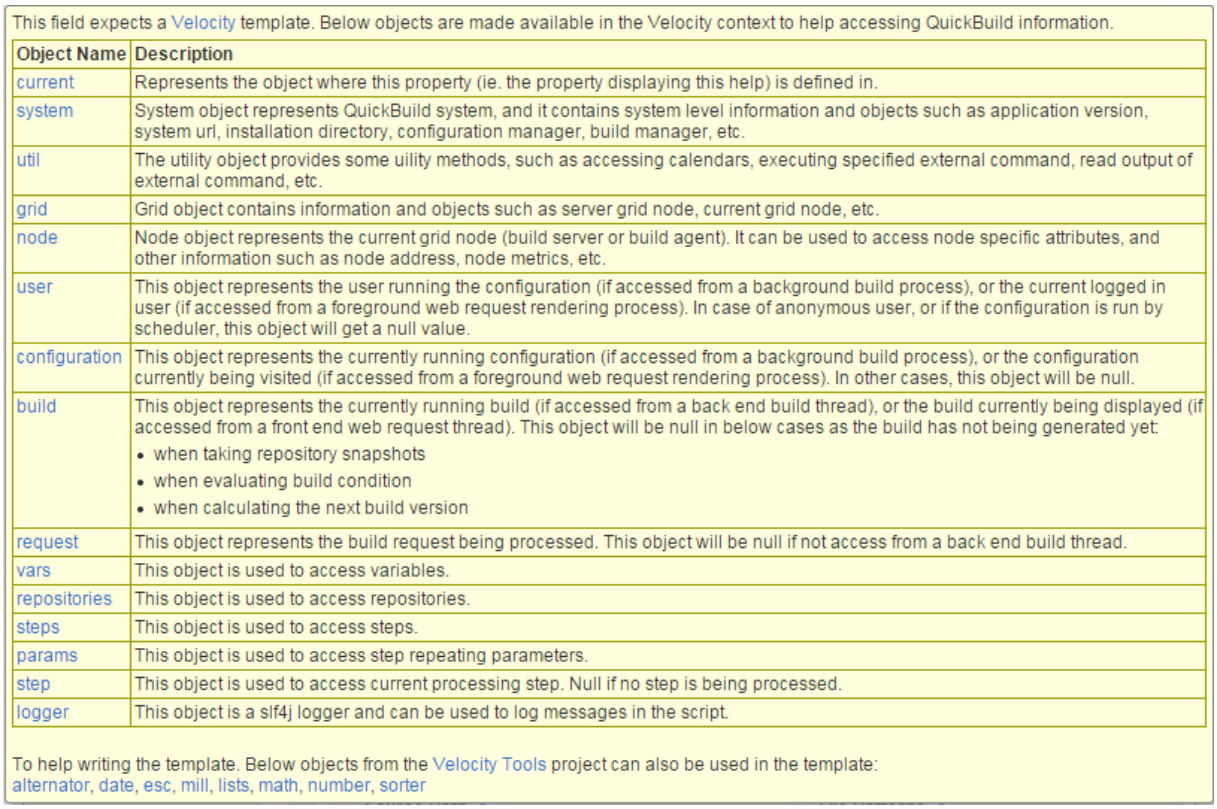
5) *{var}Link*, kus *{var}* on tagarakenduse nimi (vt 4.2.7)

6) *linkedJiraIssues* (vt 4.2.6)

7) *backendSvnChanges, frontendSvnChanges, confSvnChanges* (vt 4.2.5)

Kirja loomisel kasutab QB *Notification Template – Email Notification Template* alla lisatud Velocity malli.

Malli koostamisel kasutas autor põhjaks varasemat täieliku toote versioonimärkmeid. Kuna kasutatud kirja sisu oli HTML-is, siis oli seda lihtne muuta. Eemaldada tuli eelmises alampunktis (7.1) loendatud ainult muutuvate osade sisu, nõnda säilitades üldise struktuuri ja kujunduse. Staatiliste andmed jäeti alles, neid saab vajadusel väljasaadetavas kirjas käsitsi muuta. Kustutati üleliigsed tabeli read (*<tr></tr>*) jättes alles vaid esimese ehk tabeli päise (veerude nimed on staatilised) ja teise rea. Teise rea lahtrite täitmiseks kasutatakse Velocity mallimootorit, mille kaudu on ka võimalik ligi pääseda ka *build*'i informatsioonile, sh *vars* objektile (vt Joonis 2).



This field expects a Velocity template. Below objects are made available in the Velocity context to help accessing QuickBuild information.

Object Name	Description
current	Represents the object where this property (ie. the property displaying this help) is defined in.
system	System object represents QuickBuild system, and it contains system level information and objects such as application version, system url, installation directory, configuration manager, build manager, etc.
util	The utility object provides some uility methods, such as accessing calendars, executing specified external command, read output of external command, etc.
grid	Grid object contains information and objects such as server grid node, current grid node, etc.
node	Node object represents the current grid node (build server or build agent). It can be used to access node specific attributes, and other information such as node address, node metrics, etc.
user	This object represents the user running the configuration (if accessed from a background build process), or the current logged in user (if accessed from a foreground web request rendering process). In case of anonymous user, or if the configuration is run by scheduler, this object will get a null value.
configuration	This object represents the currently running configuration (if accessed from a background build process), or the configuration currently being visited (if accessed from a foreground web request rendering process). In other cases, this object will be null.
build	This object represents the currently running build (if accessed from a back end build thread), or the build currently being displayed (if accessed from a front end web request thread). This object will be null in below cases as the build has not being generated yet: • when taking repository snapshots • when evaluating build condition • when calculating the next build version
request	This object represents the build request being processed. This object will be null if not access from a back end build thread.
vars	This object is used to access variables.
repositories	This object is used to access repositories.
steps	This object is used to access steps.
params	This object is used to access step repeating parameters.
step	This object is used to access current processing step. Null if no step is being processed.
logger	This object is a slf4j logger and can be used to log messages in the script.

To help writing the template. Below objects from the Velocity Tools project can also be used in the template:
alternator, date, esc, mill, lists, math, number, sorter

Joonis 2. Velocity malli abiaken QB-s

Soovides ligi pääseda vars objekti muutuja *var* väärtusele, tuleb mallis kasutada *$vars.getValue("var")* (jutumärgid on kohustuslikud), mis tagastab *String*-tüüpi väärtuse. Lihtsamate andmete kirja lisamiseks sellest ka piisab, nt serveri komponendi allalaadimise link, kuid ehitamise käigus seati ka keerulisemaid välju. *LinkedJiraIssues* kasutamiseks tuleb eelnevalt selle sisu avada, st *$vars.getValue("linkedJiraIssues").split("\n")*. See käsk tagastab stringide massiivi juhul, kui sellise nimega muutujas esineb reavahetusi. Massiivid saab läbi käia *#foreach* ja muutujad väärtustada *#set* direktiivi abil. Esimene on analoogne Groovy *for*-tsükliga kasutades *in* võtmesõna.

Siinkohal tuleb kasuks Velocity Tools *lists* [23] teek, mis teeb loenditega opereerimise lihtsaks. *LinkedJiraIssues* sisu kuvatakse tabelis nii (kujunduse elemendid on eemaldatud):

```
…
#foreach ($jiraIssue in $vars.getValue("linkedJiraIssues").split("\n"))
    #set ($jiraIssueSplit = $jiraIssue.split(";;"))
    #set ($jiraKey = $lists.get($jiraIssue, 0))
    #set ($jiraUrl = $lists.get($jiraIssue, 1))
    #set ($jiraSummary = $lists.get($jiraIssue, 2))
    #set ($jiraStatus = $lists.get($jiraIssue, 3))
    #set ($jiraComment = $lists.get($jiraIssue, 4))
    #set ($jiraComponents = $lists.get($jiraIssue, 5))
    <tr>
      <td>
```

```
        <a href="$jiraUrl">$jiraKey</a>
      </td>
      <td>$jiraSummary</td>
      <td>$jiraStatus</td>
      <td>$jiraComment</td>
      <td>$jiraComponents</td>
    </tr>
#end
…
```

Hilisemates mallimootori versioonides (1.6+) on teegi kasutamine ebasoovitav, kuna sama funktsionaalsus on neis juba mootori süntaksisse integreeritud. Uuendades QB kõrgemale kui 5.1.3 versioon [24], siis on tarviklik tagantjärgi see osa mallis uuema meetodiga asendada.

Kirja, millesse on kogutud vaid konfiguratsiooni valikuväljades märgitud osad, st on modulaarne, loomiseks on kasutatud tinglauseid. Velocity on selleks *#if* ja *#ifelse* direktiivid. Tõeväärtust uuritakse *$vars.getValue("components").contains("var")* käsuga, kus *components* on string ehitatud toote osadest ja *var* kontrollitav komponent. Ümbritsedes vastava HTML osa (tabeli või selle rea) *#if* direktiiviga:

```
…
#if ($vars.getValue("componentsToBuild").contains("backend"))
    <tr>
      <td>
        <a href="$jiraUrl">$jiraKey</a>
      </td>
      <td>$jiraSummary</td>
      <td>$jiraStatus</td>
      <td>$jiraComment</td>
      <td>$jiraComponents</td>
    </tr>
#end
…
```

Kirja lõppu kuuluvat signatuuri (nimi, ametikoht, kontaktinfo) lisamine on ka automatiseeritud. Selleks kasutatakse QB konfiguratsiooni käivitava kasutaja LDAP kasutajainfot, aga kuna see ei hõlma kogu signatuuri lisatavat teavet, siis konfiguratsiooni muutujatesse on lisatud ülejäänud andmed (nt telefoni nr või Skype'i kontakti nimi).

5.3 Adresseerimine

Genereeritava kirja adressaati saab eraldi seadistada (*Notifications – Edit/Add Notification – Receivers)*. Kui ülejäänud süsteemis saadetakse kiri käivitajale, BIT-le, komponendi eest vastutava tiimi juhile ning *build*'is sisalduvate koodimuudatuste autoritele juhul, kui *build* ei õnnestu, siis versioonimärkmed saadetakse vaid kirja loomist startinud kasutajale. Nõnda jääb

vastutaval isikul veel võimalus kirja sisu kontrollida ja vajadusel muuta enne lõplikku väljasaatmist.

5.4 Silumine

Silumiseks saab lisada uue tingimuse *Notifications* alla, mille *Edit Notification* kuva *Condition* väljas on valitud „*If build is failed*“. Siis saadetakse kiri ka *build*'i ebaõnnestumisel ja kirja sisus on näha ka väärtustamata muutujad, kuna Velocity mootor prindib tühja väärtuse tagastanud *$vars.getValue(...)* käsu, mitte selle väärtuse. Üldjuhul piisab tõrke põhjusele jälile jõudmiseks konkreetse *build*'i muutujate väärtuste ja logi uurimisest.

Silumist saab teostada ka konfiguratsiooni töövoo astmete mittelubamisega või nende sisu muutmisega. Viimase meetodi puhul tuleb kasuks QB-ssse sisseehitatud objekti *logger* [25] kasutamine, mille väljund suunatakse *build*'i logisse.

Iga süsteemist tuleneva probleemi korral kuvab QB konfiguratsiooni ülevaates erindit, mis viitab asjakohasele Java *stacktrace*'ile. Niimoodi käitutakse ka juhul, kui saadetava kirja malli ei suuda Velocity Engine sõeluda või meiliserveriga on tõrge.

6. Hinnang lahendusele

Autoril kulus kogu lahenduse ja sellega seonduva välja töötamiseks ligikaudu 3-4 nädalat. Kulunud aeg oleks kindlasti olnud väiksem kui kasutada olnuks uuem versioon QB-st või sama vahend oleks samalaadsete tarkvaratoodete turul populaarsem, seetõttu oleks olnud küsimustele vastuste leidmine kiirem.

Lahendust on jooksvalt testitud nii arendamise käigus kui ka valminuna. Järk-järgulise arenduse tõttu käivitati konfiguratsiooni eri staadiumis kokku üle kahesaja korra. Valminud lahendust on seniks kasutatud vaid kahel korral, mille puhul oli konfiguratsiooni töö edukas ja puudujääke ei täheldatud. Ehkki tegemist polnud kogu toote väljalaskega, oli protsessi ajakulu vähenemine märgatav. Kui manuaalse protsessi peale võis BIT-i liikmel kuluda ligi pool tööpäevast, mil ta peab perioodiditi protsessi jälgima sõltuvate komponentide loomise alustamiseks, siis *one-click-build* võimaldab tähelepanu hoida protsessi lõpuni (või vea ilmnemiseni) muudel tööülesannetel.

Juba käivitatud konfiguratsiooni on võimalik ka peatada ja pooleli jäänud kohast jätkata, kui teha vastavasisulised valikud järgneval käivitamisel. Manuaalse töö korral aga muutujate täpsustamist ei ole, ning kasutaja saab otsejoones käivitada vajaminevate komponentide konfiguratsioonid. Sellisel juhul toob lahendus endaga kaasa lisa ajakulu uue muutujate kuva täitmise näol, kuid seda võib pidada marginaalseks, kuna tegevusele ei kulu kauem kui mõni minut - samas suurusjärgus on ekvivalentse käsitsi tehtava töö kestus.

QB võimalustest tulenevalt on võimalik lahenduse edasiarendamine - kerge on lisada uusi või muuta olemasolevaid astmeid ja muutujaid. Vajadusel saab konfiguratsiooni kopeerida ja seejärel modifitseerida nõnda loodut. Raskusi võib esineda skriptiastmetes erifunktsionaalsuse lisamisega, kuna vajalikku informatsiooni ei pruugi eksisteerida QB dokumentatsioonis.

Lahenduse loomine andis autorile kogemusi ja teadmisi kasutatava CI serveri kohta, mida saab rakendada projektides tulevikus ja olemasolevate süsteemide täiustamisel. Eriti kasulikuks on skriptimise võimekuse arendamine, kuna manusskripte saab kohandamiseks QB-s paigutada paljudesse eri väljadesse. Versioonimärkmete genereerimiseks tuli kokku puutuda mallidega, mida sai luua ja kasutada erinevatest süsteemidest (QB, JIRA,

versioonihaldus, VBPack) eraldatud andmete kuvamiseks. Sellise eri osade „liimimise“ oskuse harjutamine tuleb kasuks ka edaspidi.

7. Kokkuvõte

Agiilsete meetodite järgimine eeldab, et uus kliendikõlbulik toote versioon luuakse vähemalt iga paari nädala tagant. Keerukas toode ja samaaegselt mitu aktiivset koodiharu on asjaolud, mille tõttu rakendub olemuselt manuaalse töö lisakoormus töötajale. Koormuse kasvu tõttu kipuvad ka inimlikud eksimused sagedamini aset leidma.

Pideva integratsiooni server QuickBuild on piisavalt võimekas, et saada sellise protsessi automatiseerimisega hakkama. Seeläbi tõuseks töötaja efektiivsus, väheneks protsessile kuluv aeg ja ka vigu esineks vähem.

Töö alguses seatud eesmärk automatiseerida uue toote versiooni loomine saavutati, kuna loodud lahendus töötab ja saab oma ülesande korrektselt täidetud.

Autor lõi kasutatavas pideva integratsiooni keskkonnas konfiguratsiooni, mis nõuab töötaja tähelepanu ainult protsessi algusefaasis. Loodi muutujad, mille kaudu tekitatakse uue toote versiooni komponentide täpsustamiseks valikuväljade kuva, ning töövoog koos astmetega, mis täidaks valitud väärtuste järgi seatud ülesande. Viimasena kirjutati ka Velocity mall, mis suudaks valmistada vastava loodud versiooni komponente kajastava versioonimärkmed.

Autor on lahendust reaalsete *build request*'ide jaoks katsetanud ning pole täheldanud vigu käivitatud konfiguratsioonides ja saadetud versioonimärkmetes.

Vähenes kogu toote ehitamisele kuluv aeg, kuna üksteisele järgnevate komponentide ehitamine on vahepausideta ning versioonimärkmete välja saatmiseks kulub oluliselt vähem aega. Töötajal on *build*'i käimise ajal võimalus segamatult muude tööülesannetega tegeleda.

Tulevikus on otstarbekas loodud lahenduse sisse luua ka teste, nt ehitatud komponentide sisaldumise kontrolliks terminali või kassiiri pakis.

Summary

When practicing agile methods, one must assume that a new deliverable product version must be available every few weeks. Having a complex product and multiple active code branches at the same time result in an increase of manual labour. This, in turn, increase of human errors.

The continuous integration server QuickBuild is sufficiently capable of automating this kind of a process. This way the worker in control would have his effectivity increased, the time spent on this process decreased and also less errors would come about.

The goal set up automating the creation of a product's new version at the beginning was achieved, since the developed solution functions correctly and is able to finish the task.

The author created a configuration in VB's continuous integration environment, which requires human input only at the start. Variables were created, which were used to generate a prompt page for specifying the components needed to be built, and a workflow with steps, that are able to carry out the task according to the selected values. Lastly a Velocity template able to prepare release notes reflecting the built components was written.

The solution has been tested against real build requests and no errors have been noted in the started configurations and the release notes that have been sent out.

Moreover, the product building time decreased, since no time is spent between two sequential components and now the dispatching of release notes takes a lot less time. The worker is free to busy himself with other tasks while the build is being run.

In the future it would be reasonable to implement tests, such as a checking for built components, into the solution.

Kasutatud kirjandus

[1] Agile Alliance, „The Twelve Principles of Agile Software,“ [Võrgumaterjal]. Available: http://www.agilealliance.org/the-alliance/the-agile-manifesto/the-twelve-principles-of-agile-software/. [Kasutatud Mai 2015].

[2] scrum.ee oü, „Miks Scrum?,“ [Võrgumaterjal]. Available: http://scrum.ee/miks-scrum. [Kasutatud Mai 2015].

[3] H. Vallaste, „e-Teatmik: IT ja sidetehnika seletav sõnaraamat,“ [Võrgumaterjal]. Available: http://vallaste.ee/index.htm?Type=UserId&otsing=209. [Kasutatud Mai 2015].

[4] The Agilista PM, „An Overview of Lean-Agile Methods,“ [Võrgumaterjal]. Available: http://www.agilistapm.com/overview-of-leanagile-methods/. [Kasutatud Mai 2015].

[5] Agile Alliance, „Automated Build,“ [Võrgumaterjal]. Available: http://guide.agilealliance.org/guide/autobuild.html. [Kasutatud Mai 2015].

[6] Playtech Estonia OÜ, [Võrgumaterjal]. Available: http://www.playtech.ee/?nav=playtechgroup_ee.

[7] Jenkins CI, „Jenkins,“ [Võrgumaterjal]. Available: https://jenkins-ci.org/. [Kasutatud April 2015].

[8] JetBrains s.r.o, „TeamCity,“ [Võrgumaterjal]. Available: https://www.jetbrains.com/teamcity/. [Kasutatud April 2015].

[9] Atlassian, „Continuous Integration & Build Server - Bamboo,“ [Võrgumaterjal]. Available: https://www.atlassian.com/software/bamboo. [Kasutatud April 2015].

[10] PMEase Incorporated, „PMEase - Continuous integration and deployment solution!,“ [Võrgumaterjal]. Available: http://www.pmease.com/. [Kasutatud April 2015].

[11] PMEase Inc, „Configuration,“ [Võrgumaterjal]. Available: http://wiki.pmease.com/display/QB40/Configuration. [Kasutatud Mai 2015].

[12] Atlassian, „What is an Issue?,“ [Võrgumaterjal]. Available: https://confluence.atlassian.com/display/JIRA/What+is+an+Issue.

[13] Atlassian, „JIRA Agile,“ [Võrgumaterjal]. Available: https://www.atlassian.com/software/jira/agile.

[14] Atlassian, „Atlassian Blogs,“ [Võrgumaterjal]. Available: http://blogs.atlassian.com/2014/02/evaluating-jira-agile-team-members/. [Kasutatud Mai 2015].

[15] The Apache Software Foundation, „Apache Velocity Site - Velocity Engine,“ [Võrgumaterjal]. Available: http://velocity.apache.org/engine/. [Kasutatud Mai 2015].

[16] The Apache Software Foundation, „Maven - Settings Reference,“ [Võrgumaterjal]. Available: https://maven.apache.org/settings.html. [Kasutatud Mai 2015].

[17] The Apache Software Foundation, „Maven - POM Reference,“ [Võrgumaterjal]. Available: https://maven.apache.org/pom.html#Maven_Coordinates. [Kasutatud Mai 2015].

[18] PMEase, „Chained Build Options,“ [Võrgumaterjal]. Available: http://wiki.pmease.com/display/QB40/Chained+Build+Options. [Kasutatud Mai 2015].

[19] PMEase Inc, „Promote Build,“ [Võrgumaterjal]. Available:

http://wiki.pmease.com/display/QB40/Promote+Build. [Kasutatud April 2015].

[20] B. W. F. a. C. M. P. Ben Collins-Sussman, „svn log,“ [Võrgumaterjal]. Available: http://svnbook.red-bean.com/en/1.7/svn.ref.svn.c.log.html. [Kasutatud Mai 2015].

[21] Atlassian, „JIRA 6.4.4 REST API documentation,“ [Võrgumaterjal]. Available: https://docs.atlassian.com/jira/REST/latest/#d2e965.

[22] Sourceforge, „Maven - Json-lib,“ [Võrgumaterjal]. Available: http://json-lib.sourceforge.net/. [Kasutatud Mai 2015].

[23] Apache Software Foundation, „ListTool (VelocityTools 2.0-beta4 documentation),“ [Võrgumaterjal]. Available: http://velocity.apache.org/tools/devel/javadoc/org/apache/velocity/tools/generic/ListTool.html. [Kasutatud Mai 2015].

[24] PMEase, „[#QB-1856] Please upgrade Velocity to the latest version,“ [Võrgumaterjal]. Available: http://track.pmease.com/browse/QB-1856.

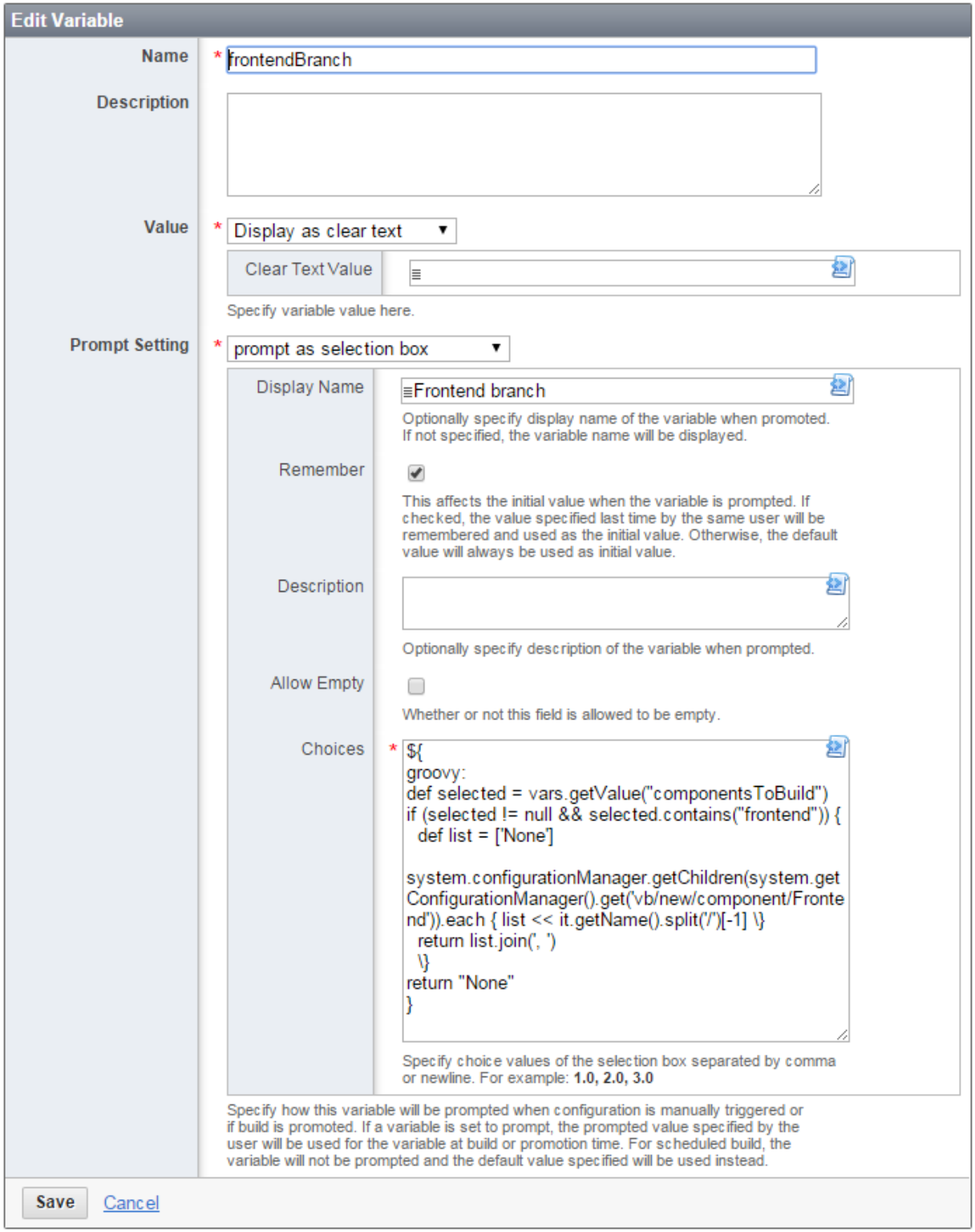
[25] QOS.ch, „Logger (SLF4J 1.7.12 API),“ [Võrgumaterjal]. Available: http://www.slf4j.org/apidocs/org/slf4j/Logger.html. [Kasutatud Mai 2015].

[26] PMEase Inc, „Repository,“ [Võrgumaterjal]. Available: http://wiki.pmease.com/display/QB40/Repository. [Kasutatud April 2015].

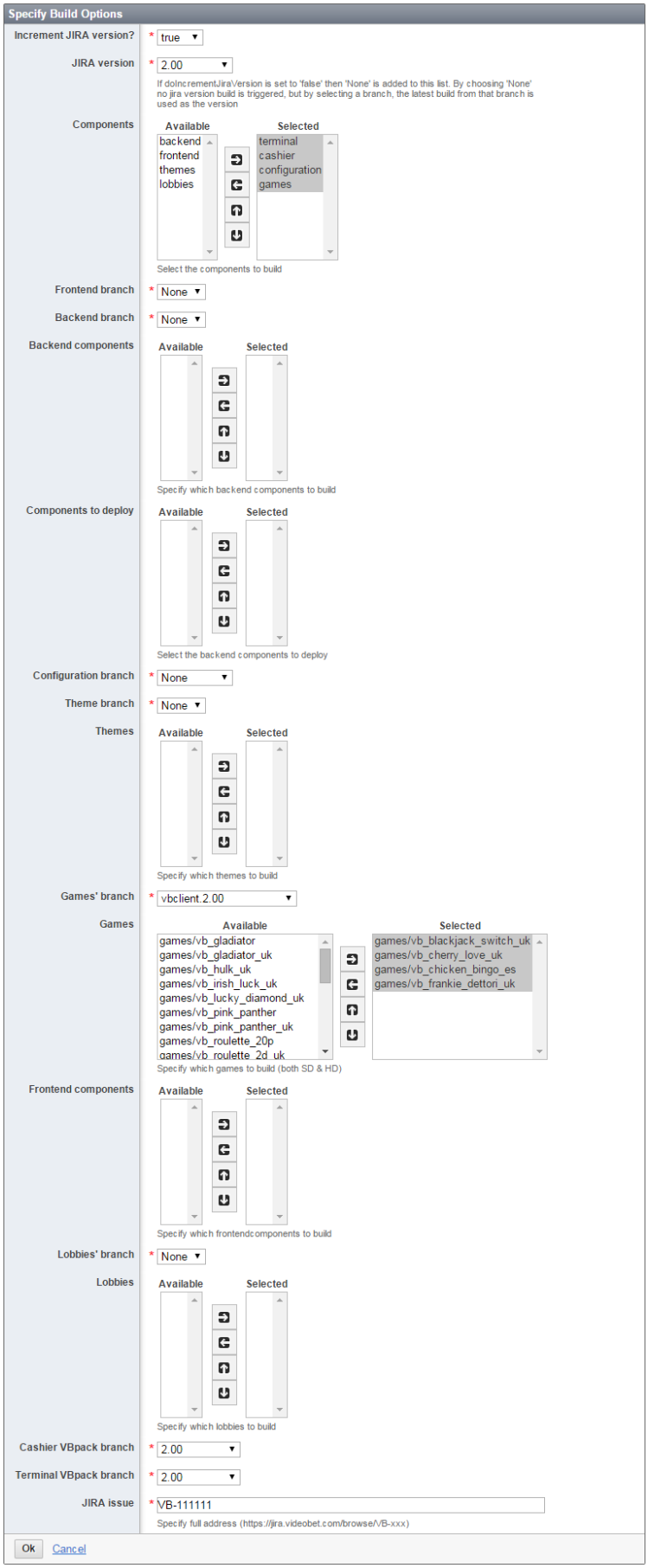
[27] M. Fowler, „Continuous Integration,“ [Võrgumaterjal]. Available: http://www.martinfowler.com/articles/continuousIntegration.html. [Kasutatud April 2015].

Lisa 1

Joonis 3. Muutuja *frontendBranch* redigeerimisleht



Joonis 4. Build'i käivitamisel kuvatavad valikuväljad



Lisa 2

Tabel 1. Loodud konfiguratsiooni muutujad

Nr	Nimi	Tüüp	Valikud (komaga eraldatud)
1.	doIncrementVersion	*Selection*	*true, false*
2.	jiraVersion	*Selection*	Kõik *vb/new/component/release_jira* alamkonfiguratsioonide nimed. Kui doIncrementVersion (1.) on *false*, siis lisatakse valikusse veel *None*
3.	availableComponents	Eelväärtustatud	*terminal, cashier, backend, frontend, configuration, themes, lobbies, games*
4.	componentsToBuild	*Multi-selection*	availableComponents (3.) sisu kuvatud valikutena
5.	frontendBranch	*Selection*	Kõik *vb/new/component/Frontend* alamkonfiguratsioonide nimed ja *None*. Ainult *None* kui componentsToBuild (4.) ei sisalda *frontend*'i
6.	frontendComponents	*Multi-selection*	Tühi kui valikut pole frontendBranch (5.) muutujas tehtud, muidu *'vb/new/component/Frontend/' + vars.getValue('frontendBranch') + '/binaries'* teega alamkonfiguratsioonide nimed.
6.	backendBranch	*Selection*	Kõik *vb/new/component/Backend* alamkonfiguratsioonide nimed ja *None*. Ainult *None* kui componentsToBuild (4.) ei sisalda *backend*'i

7.	backendComponents	*Multi-selection*	Tühi kui valikut pole backendBranch (6.) muutujas tehtud, muidu *'vb/new/component/Backend/' + vars.getValue('backendBranch')* teega vahetute alamkonfiguratsioonide nimed (v.a *server*). Lisaks samas stiilis *'vb/new/component/Backend/' + vars.getValue('backendBranch') + '/server'* alt (proksid)
8.	deployComponents	*Multi-selection*	Tühi kui valikut pole backendBranch (6.) muutujas tehtud, muidu kõik juurutamise käivitamiseks mõeldud konfiguratsioonide teed (s.t on *deploy*-nimelise vahetu alamkonfiguratsiooniga) *'vb/new/component/Backend/' + vars.getValue('backendBranch')* alt
9.	confBranch	*Selection*	Kõik *vb/new/component/configuration* alamkonfiguratsioonide nimed ja *None*. Ainult *None* kui componentsToBuild (4.) ei sisalda *configuration*'it
10.	themeBranch	*Selection*	Kõik *vb/new/component/content* alamkonfiguratsioonide nimed ja *None*. Ainult *None* kui componentsToBuild (4.) ei sisalda *configuration*'it
11.	themesToBuild	*Multi-selection*	Tühi kui valikut pole frontendBranch (5.) muutujas tehtud, muidu *'vb/new/component/Frontend/' + vars.getValue('frontendBranch') + '/binaries'* teega alamkonfiguratsioonide nimed
12.	gameBranch	*Selection*	Kõik *vb/new/component/content* alamkonfiguratsioonide nimed ja *None*. Ainult *None* kui componentsToBuild (4.) ei sisalda *game*'i
13.	gamesToBuild	*Multi-selection*	Tühi kui valikut pole gameBranch (12.) muutujas tehtud, muidu nii *'vb/new/component/content/' + vars.getValue('gameBranch') + '/games'* või *'/games_hd'* teega alamkonfiguratsioonide nimed
14.	lobbyBranch	*Selection*	Kõik *vb/new/component/content* alamkonfiguratsioonide nimed ja *None*. Ainult *None* kui componentsToBuild (4.) ei sisalda *lobby*

15.	lobbiesToBuild	*Multi-selection*	Tühi kui valikut pole lobbyBranch (12.) muutujas tehtud, muidu nii *'vb/new/component/content/' + vars.getValue('lobbyBranch') + '/lobbies'* teega alamkonfiguratsioonide nimed
16.	cashierVbpackBranch	*Selection*	Kõik *vb/new/component/content* alamkonfiguratsioonide nimed ja *None*. Ainult *None* kui componentsToBuild (4.) ei sisalda *lobby*
17.	terminalVbpackBranch	*Selection*	Kõik *vb/new/component/content* alamkonfiguratsioonide nimed ja *None*. Ainult *None* kui componentsToBuild (4.) ei sisalda *lobby*
18.	jiraIssue	*Text*	JIRA *build request* lingina või JIRA probleemi identifikaator, kujul *VB-xxx*

Järgnevas tabelis on välja toodud loodud konfiguratsiooni kõik astmed. Käivitamistingimuse veerus vastab astme *Execute Condition* väljale, kus saab kasutada Groovy koodi, mille väärtus peab olema tõene (s.t *if (*condition == *true*), kus *condition* on vastava rea „Käivitamistingimuse“ veeru väärtus), et astet läbida. Astmed moodustavad puu struktuuri, mille täpse kujuni jõuab kasutades *master* astet juurena ja „Otsesed alamastmed“ veerust sõlme vahetu järglase.

Tabel 2. Konfiguratsiooni töövoos kasutatud astmed ja nende paiknemine

Nr	Nimi	Tüüp	Käivitamistingimus	Otsesed alamastmed
0.	master	*Sequential*	-	1.-3., 14., 21., 28.-34.
1.	trigger-jira-version-build	*Trigger Other Builds*	`vars.getValue('doIncrementVersion') == 'true' && vars.getValue('jiraVersion') != 'None'`	-

2.	set-build-version	*Execute Script*	*Always*	-
3.	trigger-parallel-backend-frontend-configuration-builds	*Parallel*	-	4., 5., 10.
4.	trigger-configuration-build	*Trigger Other Builds*	`vars.getValue('componentsToBuild').contains('configuration') && vars.getValue('confBranch') != 'None'`	-
5.	trigger-backend-build	*Sequential*	`vars.getValue('componentsToBuild').contains('backend') && vars.getValue('backendBranch') != ’None’`	6., 8.
6.	trigger-parallel-backend-build	*Parallel*	`vars.getValue('componentsToBuild').contains('backend') && !vars.getValue('deployComponents')?.isEmpty()`	7.
7.	trigger-backend-component-build	*Trigger Other Builds*	*If all previous sibling steps are successful*	-
8.	trigger-parallel-backend-deploy	*Parallel*	`vars.getValue('componentsToBuild').contains('backend') && !vars.getValue('deployComponents')?.isEmpty()`	9.
9.	trigger-backend-component-deploy	*Trigger Other Builds*	*If all previous sibling steps are successful*	-

10.	trigger-frontend-build	*Sequential*	`vars.getValue('componentsToBuild').contains('frontend') && vars.getValue('frontendBranch') != 'None'`	11., 13.
11.	trigger-sequential-frontend-binaries-build	*Sequential*	-	12.
12.	trigger-frontend-binaries-build	*Trigger Other Builds*	*If all previous sibling steps are successful*	-
13.	trigger-frontend-gam-build	*Trigger Other Builds*	*If all previous sibling steps are successful*	-
14.	trigger-parallel-content-builds	*Parallel*	*If all previous sibling steps are successful*	15., 16., 19.
15.	trigger-parallel-themes-build	*Parallel*	`vars.getValue('componentsToBuild').contains('themes') && vars.getValue('themeBranch') != 'None'`	16.
16.	trigger-theme-build	*Trigger Other Builds*	*If all previous sibling steps are successful*	-
17.	trigger-parallel-lobbies-build	*Parallel*	`vars.getValue('componentsToBuild').contains('lobbies') && vars.getValue('lobbyBranch') != 'None'`	18.
18.	trigger-lobby-build	*Trigger Other Builds*	*If all previous sibling steps are successful*	-

19.	trigger-parallel-games-build	*Parallel*	`vars.getValue('componentsToBuild').contains('games') && vars.getValue('gameBranch'`	20.
20.	trigger-game-build	*Trigger Other Builds*	*If all previous sibling steps are successful*	-
21.	trigger-parallel-vbpack-builds	*Parallel*	*If all previous sibling steps are successful*	22.,
22.	trigger-sequential-cashier-vbpack-and-promotion-build	*Sequential*	`vars.getValue('componentsToBuild').contains(step.name.split('-')[1]) && vars.getValue(step.name.split('-')[1] + 'VbpackBranch') != 'None'`	23., 24.
23.	trigger-cashier-vbpack-build	*Trigger Other Builds*	*If all previous sibling steps are successful*	-
24.	trigger-cashier-vbpack-promotion	*Execute Script*	*If all previous sibling steps are successful*	-
25.	trigger-sequential-terminal-vbpack-and-promotion-build	*Sequential*	`vars.getValue('componentsToBuild').contains(step.name.split('-')[1]) && vars.getValue(step.name.split('-')[1] + 'VbpackBranch') != 'None'`	26., 27.

26.	trigger-terminal-vbpack-build	*Trigger Other Builds*	*If all previous sibling steps are successful*	-
27.	trigger-terminal-vbpack-promotion	*Execute Script*	*If all previous sibling steps are successful*	-
28.	checkout-release_note	*Checkout*	*If all previous sibling steps are successful*	-
29.	publish-email-image	*Artifacts*	*If all previous sibling steps are successful*	-
30.	get-svn-logs	*Execute Script*	*If all previous sibling steps are successful*	-
31.	get-revisions	*Execute Script*	*If all previous sibling steps are successful*	-
32.	get-jira-linked-issues	*Execute Script*	*If all previous sibling steps are successful*	-
33.	set-user-info	*Execute Script*	*If all previous sibling steps are successful*	-
34.	get-backend-links	*Execute Script*	`vars.getValue('componentsToBuild').contains('backend') && vars.getValue('backendBranch') != 'None'`	-

Lisa 3

Koodinäide 1. Terminali/kassiiri ülendamine koos tegevuse lõpu ootamisega

groovy:

```
// this step needs to be run on _server_
// courtesy of http://forum.pmease.com/viewtopic.php?f=1&t=2309

import com.pmease.quickbuild.*
import com.pmease.quickbuild.persistence.*

def confName = step.name.split('-')[1] // cashier or terminal
def confToPromote = system.configurationManager.get("vb/new/component/release/" +
vars.getValue(confName + 'VbpackBranch') + "/" + confName)
def buildToPromote = confToPromote.latestBuild
def promotionDef =
ScriptEngine.instance.installInterpolator(confToPromote.findPromotion("Promote to
QA"))

Context.push(buildToPromote)
try {
  promotionDef.promote(buildToPromote, [:])
} finally {
  Context.pop()
}

SessionManager.openSession()
try {
      while (!finished) {
            sleep(1000)
            buildToPromote = system.buildManager.load(buildToPromote.id)
            if (buildToPromote.promotedTo.isEmpty())
              continue
            def promotedBuild = buildToPromote.promotedTo.get(0)
            if (promotedBuild.isRunning())
              continue
            else if (promotedBuild.isFailed()) {
              vars.get(vars.getValue(confName + "VbpackLink")).setValue("")
              throw new QuickbuildException("Promote build is failed.")
            }
            else { // build finished
              def artifactName = confName + '-' + promotedBuild.getVersion() +
'.vbpack'
              vars.get(vars.getValue(confName +
"VbpackLink")).setValue(build.getDownloadUrl('artifacts/' + artifactName, true))
              break
            }
      }
} finally {
  SessionManager.closeSession()
```

```
}
```

Koodinäide 2. VBPack'i põhikomponentide muudatuse versiooni hankimine

```
groovy:
…
def getComponentRevision(coreCoordinates, termOrCashConf) {
    logger.info("confMap: {}", confMap)
    def artifactAndCoordinates = []
    for (coord in coreCoordinates) { // check all 'Core' coordinates
        def artifactId = coord.split(':')[1] // get their artifactId
        def artifactVersion = coord.split(':')[-1]
        // find the configuration path for this artifact
        logger.info("artifactId: {}", artifactId)
        def found = confMap.find { path, components ->
            checkComponentListForArtifact(components, artifactId)
        }
        def qbConfName = getQBConfName(found.value, artifactId, termOrCashConf)
        def qbConfBase = found.key
        if (found.key.contains('None')) {
            qbConfBase = determineQBConfFromVersion(found.key, artifactId,
artifactVersion, qbConfName)
        }
        def confToSeek = qbConfBase + qbConfName
        logger.info("confToSeek: {}", confToSeek)
        def builds =
system.buildManager.getBuilds(system.configurationManager.get(confToSeek))
        def foundBuild = builds.find { build ->
!build.getVersion().contains('SNAPSHOT') &&
build.getVersion().contains(artifactVersion) }
        if (!foundBuild) { logger.warn("Couldn't find build for " + coord + " in "
+ found.key); continue }
        def revision = foundBuild.getRepositories().find { repo ->
!repo.getName().contains('vb.build.tools.maven') && repo.isCheckout()
}.getRevision().toString()
        logger.info(coord + "@" + revision)
        artifactAndCoordinates << coord + "@" + revision
    }
    return artifactAndCoordinates
}

/**
* Takes a configuration path with 'None' string in it, extracts the base path
which it uses to look for children configurations.
* The children configuration names will be used to search for the build which
version is in the build name.
* If found, will return the path to the configuration of which only the last part
is missing.
*/
def determineQBConfFromVersion(confPathWithNone, artifactId, artifactVersion,
qbConfName) {
    def confPathBase = confPathWithNone.split('None')[0]
    for (parentConf in
system.configurationManager.getChildren(system.configurationManager.get(confPathBa
se)).reverse()) {
        def parentName = parentConf.getName()
        def artifactConf =
system.configurationManager.get(confPathWithNone.replace('None', parentName) +
qbConfName)
```

```
        if (system.buildManager.getBuilds(artifactConf).find { build ->
!build.getVersion().contains('SNAPSHOT') &&
build.getVersion().contains(artifactVersion) })
            return confPathWithNone.replace('None', parentName)
    }
    return confPathBase
}

/**
* Gets the configuration name from given component list (backend, frontend, etc.).
* Processes %conf% by replacing it with conf (terminal/cashier) string.
*/
def getQBConfName(componentList, artifactId, conf) {
    def qbConfName = componentList.find { component ->
component.contains(artifactId) }
    qbConfName = qbConfName.replaceAll("%conf%", conf)
    return qbConfName.contains(':') ? qbConfName.split(':')[-1] : qbConfName
}

def checkComponentListForArtifact(componentList, artifactId) {
    return componentList.any { it.split(':')[0] == artifactId  }
}

/**
*  Gets contents of pom.xml which is contained in (terminal/cashier) vbpack
*/
def getVbpackPomTextFromBuild(build) {
    def confName = build.getConfiguration().getName() // either 'cashier' or
'terminal'
    def artifactName = confName + '-' + build.getVersion() + '.vbpack'
    def vbpackName = downloadVbpack(build.getDownloadUrl('artifacts/' +
artifactName, true))
    def vbpack = new File(vbpackName)
    def zipFile = new ZipFile(vbpack)
    String pomText = ""
    zipFile.entries().each {
        if (it.getName().endsWith("pom.xml"))
            pomText = zipFile.getInputStream(it).text
    }
    if (pomtText?.size() > 0) {
        throw new com.pmease.quickbuild.QuickbuildException(vbpackName + " doesn't
contain pom.xml or the pom.xml file is empty!")
    }
    return pomText
}
…
```

Koodinäide 3. JIRA alamprobleemide päring ja sõelumine läbi REST rakendusliidese

groovy:

```
import net.sf.*
import net.sf.json.*
import net.sf.json.groovy.*

GJson.enhanceString()

def jiraIssue = (vars.getValue('jiraIssue') =~ /VB\-\d+/)[0]
def restURL = getAuthenticatedRestUrl(jiraIssue, "", "GET")
```

```
def responseCode = restURL.getResponseCode()

if (responseCode != 200) {
    logger.warn("{} GET query returned with {}: {}!", vars.getValue('jiraIssue'),
responseCode, restURL.getResponseMessage())
    return
}

def resultText = restURL.getInputStream().getText()
def resultAsJsonObject = resultText as JSONObject
def jiraBase = "https://jira.videobet.com/browse/"

vars.get("jiraBuildIssueLink").setValue(jiraBase + jiraIssue)

def jiraIssues = []
for (issue in resultAsJsonObject["issues"]) {
    def key = issue["key"]
    def url = jiraBase + ji.key
    def summary = issue["fields"]["summary"]
    def status = issue["fields"]["status"]["name"]
    def components = []
    for (component in issue["fields"]["components"]["name"]) {
        components << component
    }
    def components = components.join(", ")
    def comment = issue["fields"]["comment"] != null || issue["fields"]["comment"]
== "null" ? " " : issue["fields"]["comment"]
    def jIssue = ji.key + ";;" + ji.url + ";;" + ji.summary + ";;" + ji.status +
";;" + ji.comment + ";;" + ji.components
    jiraIssues << jIssue
}

vars.get("linkedJiraIssues").setValue(jiraIssues.join("\n"))
def getAuthenticatedRestUrl(String issue, String parameter, String method) {
    def restURL =
('https://jira.videobet.com/rest/api/2/search?jql=issue+in+linkedIssues("' + issue
+
'")+order+by+status&maxResults=500&fields=key,summary,status,comment,components').
toURL().openConnection()
    // JIRA needs auth
    def remoteAuth = "Basic ZHVtbXk6ZHVtbXk=" //
vars.getValue("QBPassword")).bytes.encodeBase64().toString()
    restURL.setRequestProperty("Authorization", remoteAuth)
    restURL.setRequestMethod(method)
    return restURL
}

def doPostJsonRequest(URLConnection restURL, String json) {
    restURL.setDoOutput(true) // false by default
    restURL.setRequestProperty("Content-Type", "application/json")
    restURL.outputStream.withWriter { Writer writer ->
        writer << json
    }
    logger.info("responseCode: " + restURL.responseCode)
    if (restURL.responseCode != 204)
        logger.warn("Transitioning JIRA issue returned {}", restURL.responseCode)
    return restURL.responseCode
}
```