TARTU ÜLIKOOL
Arvutiteaduse instituut
Informaatika õppekava

Karl Soosalu

Kvaliteedi tagamise protsessid tarkvaraettevõttes Playtech Estonia näitel

Bakalaureusetöö (9 EAP)

Juhendajad: Heili Orav, *PhD*
Jorma Pärn

Tartu 2022

Kvaliteedi tagamise protsessid tarkvaraettevõttes Playtech Estonia näitel

Lühikokkuvõte:

Kvaliteedi tagamine ja testimine on tarkvara arendustsükli olulised osad, mis mõjutavad suurel määral projektide edukust. Suur roll on nii funktsionaalsetel kui ka mittefunktsionaalsetel nõuetel, mis mõlemad klientide kasutuskogemust suurel määral mõjutavad. Käesoleva bakalaureusetöö eesmärgiks oli analüüsida, kuidas tagada toote kvaliteet tänapäevases tarkvaraettevõttes. Samuti selgitada välja, milliseid testimisviise on olemas, kui suurt osa on võimalik automatiseerida ning milline mõju on kaasnenud välearenduse tulekuga.

Kvaliteedi tagamise protsesse kirjeldatakse Playtech Estonia OÜ osakonna näitel. Selleks uuriti ettevõtte tööde kulgu ning intervjueeriti töötajaid, selgitati välja nende teadmised ja suhtumine testimise erinevatesse meetoditesse. Lõpuks tehti töös parandusettepanekud edukamaks kvaliteedi tagamiseks uuritud osakonnas. Selgus, et testimine on ettevõtte tööprotsessis olulisel kohal ning lisaks automatiseeritud testidele kasutatakse palju ka käsitsi testimist. Suurim probleem on senimaani olnud jõudluse testimisega, mida on testkeskkonnas rakendatud minimaalselt. Töö tulemusena tehtud parandusettepanekud viitavad jõudluse testimise edendamisele, üksuste testimise osakaalu suurendamisele, kvaliteedi kontrolli veelgi suuremale automatiseerimisele ning dokumentatsiooni arendamisele.

Võtmesõnad:

Informaatika, kvaliteedi tagamine

CERCS: P175 Informaatika, süsteemiteooria

Quality Assurance Processes in a Software Company on the Example of Playtech Estonia

Abstract:

Quality assurance and testing are important parts of the software development cycle that highly affect the success of projects. Both functional and non-functional requirements have a big impact and affect the satisfaction of the client to a large extent. This Bachelor's thesis was intended to analyse the quality assurance processes in today's software company. Also, to find out what kind of testing methods are available, how much can be automated, and how agile development has changed the field.

The processes of quality assurance were described on the example of Playtech Estonia's department. For that purpose, the working structure of the company was examined, and employees were interviewed to find out their knowledge and attitudes toward different testing methods. Finally, the author made suggestions for improvements to ensure a more successful quality assurance process. The results showed that testing is an important part of the work process inside the company and manual testing is widely used besides automated tests. The biggest problem so far has been performance testing, which has basically remained untested in test environments. Proposed improvements suggest that performance testing must be enhanced, documentation should be improved, the importance of unit testing could be increased, and automated tests must be used even more.

Keywords:

Informatics, Quality Assurance

CERCS: P175 Informatics, systems theory

Sisukord

Sissejuhatus

Tarkvara loomine on protsess, mis jaguneb mitmeks etapiks ning oodatav lõpptulemus saavutatakse, kui need kõik edukalt täidetakse. Kvaliteedi kontroll ja testimine on selle olulised osad, mis aitavad kindlustada vastavuse nõutud standarditele. Tarkvaraarendus on viimastel kümnenditel pidevalt ja suurel määral arenenud ning koos sellega on muutunud ka kvaliteedi tagamise protsessid. Tarkvara välearenduse (ingl *agile development*) meetodid on kaasaaegsetes ettevõtetes väga levinud, võimaldades kiireid muutusi ja kliendi vajaduste efektiivset rahuldamist. Sellistes kiirelt ümber kujunevates tarkvara projektides on kvaliteedi tagamine keeruline, kuid ülimalt oluline väljakutse, mis tihtipeale võib määrata projektide või kogu ettevõtte edukuse. Testimisprotsesse võivad mõjutada väga mitmed aspektid ning seega on oluline olla nendest teadlik ning mõista, kuidas praktikas teenuse kvaliteeti efektiivselt kontrollida ja tõsta.

Töö eesmärk on analüüsida, kuidas tagada toote kvaliteet tänapäevases tarkvaraettevõttes. Samuti selgitada välja, milliseid testimisviise on olemas ning kui suurt osa on võimalik automatiseerida. Uuritakse, millisel määral mõjutavad välearenduse põhimõtted testimist ning kui olulised on traditsioonilised testimismeetodid. Kvaliteedi tagamise protsesse kirjeldatakse osaühing Playtech Estonia osakonna näitel. Selleks vaadeldakse tööde kulgu ning intervjueeritakse töötajaid, selgitades välja nende teadmised ja suhtumine erinevatesse testimise meetoditesse. Lõpuks tehakse uuritud osakonnale parandusettepanekud edukamaks kvaliteedi tagamiseks.

Käesoleva bakalaureusetöö esimeses peatükis antakse teoreetiline ülevaade erinevatest meetoditest, millega tarkvara toimimist tagada. Kirjeldatakse tuntumaid töövõtteid ja selgitatakse välja, millele oleks vaja nii funktsionaalse kui ka mittefunktsionaalse poole pealt tähelepanu pöörata. Töö teine pool kajastab Playtech Estonias läbi viidud praktilist uurimist. See algab töö metoodika kirjeldusega, milles sisaldub autori iseseisva uurimise seletus ettevõttes ning selgitused küsimustiku koostamise, valimi moodustamise ning intervjuude läbiviimise kohta. Edasi arutletakse saadud tulemuste üle, kirjeldatakse olukorda ettevõttes kõrvaltvaataja perspektiivist ja asjaga kursis olevate inimeste intervjuudest kuuldu põhjal. Viimases alapeatükis tehakse ettepanekuid, mida vastavalt välearenduse lähenemistele ning tarkvara testimise headele tavadele oleks võimalik parandada.

1. Kvaliteedi tagamise protsessid

Inimkonna arengu ja infoühiskonna pealetungi käigus on tarkvara loomise protsessid olnud pidevas muutuses. Tänapäeval on selgelt levinuimaks metoodikaks kujunenud välearendus. See põhineb tänaseni peamiselt tarkvara välearenduse manifestil [1], mis rõhutab nelja põhimõtet:

1) Inimeste omavaheline suhtlus on tähtsam kui arendusvahendid ja protsessid.
2) Töötav tarkvara on olulisem kui dokumentatsioon.
3) Võrreldes lepinguliste läbirääkimistega on koostöö kliendiga prioriteetsem.
4) Ümberkorraldused tarkvaraprojekti varajases faasis on efektiivsemad kui algse plaani järgimine.

Kvaliteedi tagamine tarkvaraettevõtetes on muutunud manifestile sarnaselt ning on ajaga palju arenenud. Edenemine on toimunud nii iseseisvalt uute tööriistade ja lahenduste näol kui ka koos välearenduse võtete laialdase levikuga. Kui minevikus olid ettevõtetes laialdaselt kasutuses näiteks testiplaanid ja kvaliteedikontroll, mis toimus tihtipeale tarkvaraprotsessi lõpufaasis, siis tänapäevastes välearenduse meetodites on selline lähenemine pigem kadunud. Paljudes olukordades mõjub välearendus lisaks ka kvaliteedile positiivselt ning viimase "State of Agile report" [2] kohaselt valib lausa 42 protsenti ettevõtetest välearenduse, kuna see aitab tõhustada tarkvara kvaliteeti tagamist.

Pidevalt muutuva tootega on testijad kogu arendustsükli vältel töös ja kvaliteet peab olema erinevate toote versioonidega tagatud pidevalt. See nõuab mitmete osapoolte kommunikatsiooni ja koostööd ning oluline on klientide tundmine ja selgelt kirjas olevad nõuded. Lisaks on testija rollid hakanud teiste ametitega segunema, kuna arendajad vastutavad kvaliteedi eest, kirjutades ise rohkem teste ning kvaliteedi tagamise insenerid ei otsi ainult vigu, vaid lisaks ka puuduvaid funktsionaalsusi ja võimalusi, kuidas tarkvara üldiselt paremaks muuta [3]. Erinevaid kvaliteedi tagamise võimalusi on mitmeid ning traditsioonilisemad testimise vahendid on segunenud kaasaegsematega, kuid üldiselt saab vajalikud toimingud liigitada funktsionaalseteks ja mittefunktsionaalseteks meetoditeks.

1.1. Funktsionaalne testimine

Funktsionaalne testimine tähendab süsteemi kindla osa toimimise kontrollimist. Nõuded on tavaliselt ette antud ja dokumenteeritud ning nende põhjal tuleb veenduda, kas kõik on kasutajate ootustele vastav. Funktsionaalse testimise saab jagada testitava komponendi suurusele vastavalt süsteemi, integratsiooni ja üksuste testimiseks (ingl *unit testing*). Üksuste testimine keskendub spetsiifiliste funktsioonide või klasside töö kontrollimisele, millega veendutakse, et kõik lisatud meetodid teevad seda, mille jaoks need loodi. Integratsiooni testid vaatavad üle, et rakenduse erinevad osad koos toimiksid ning süsteemi testimine valideerib lõpliku toote korrektse töötamise. Üksuste testimine on ettevõtete lõikes küllaltki sarnane ning toimub üldjuhul protsessi varajases staadiumis. Seevastu integratsiooni ja süsteemi testimine võib toimuda projekti erinevates etappides ning nende käigus võib kaasatud olla nii käsitsi kui ka automatiseeritud testimine. Üksuste ja automatiseeritud testimiste juures on olemas omad eripärad, mida kvaliteedi tagamisel on võimalik efektiivselt ära kasutada. Järgnevalt neid kirjeldataksegi.

1.1.1. Üksuste testimine

Väiksemamahuliste süsteemi osade ehk üksuste testimine jääb üldiselt arendajate vastutusalasse ning asub rakendusega samas koodibaasis. See annab lisaks kvaliteedi kontrollimisele lihtsa võimaluse jälgida, millised osad koodist on testide poolt kontrollitud. Antud omadust saab iseloomustada protsendina, mis näitab, kui suur osa koodist testide käigus käivitatakse. Erinevaid mõõtmisviise on mitmeid. Vaadelda saab, kui suurt osa funktsioonidest jooksutatakse, millise protsendi lauseteni (ingl *statement*) jõutakse või millised võimalused vaadeldakse läbi harudes (ingl *branch*) [4]. Neist levinuim ja lihtsaim on lausekate (ingl *statement coverage*). Kõrge protsent ei kindlusta küll olukorda, kus defekte ei esine, kuid madal number väljendab tihtipeale süsteemi haavatavust erinevatele võimalikele probleemidele. Vajalik katvus sõltub paljudest asjaoludest ja kindlat numbrit ei eksisteeri, kuid on levinud arvamus, et 70 protsendi juurde jääv ja sellest kõrgem testidega kaetus on üldjuhul piisav [5]. Üksuste testimise juures on enamikel juhtudel soovituslik automatiseerimine ehk uute muudatuste korral võiks süsteem ise kontrollida, et eelnevad testid toimima jääksid.

1.1.2. Automatiseeritud testimine

Testimise ning eriti funktsionaalse kvaliteedi kontrolli juures kerkib tihtipeale küsimus, mida saaks automatiseerida ja mida peab üle vaatama käsitsi. Automaatselt käivituvatel testidel on mitmed selged eelised. Esiteks säästab see inimeste aega, kuna sama tegevust ei ole vaja pidevalt uuesti korrata. See tähendab, et automatiseerimine aitab garanteerida rakenduse korrektse toimimise ka pärast uuendusi ehk regressioontestid on edukad. Seega ei ole vaja toimivaid süsteemi osasid uuesti üle testida ning lisaks on sõltuvalt tootest võimalusi automatiseeritud testide edukaks rakendamiseks kindlasti palju enam [6]. Samuti annab automatiseeritud protsess toote kohta kiiresti tagasisidet ideaalis juba arenduse käigus ehk vead leitakse välearenduslikku lähenemist soosivalt üles väga varajases staadiumis. Kõik see võimaldab üles ehitada pidevintegratsiooni (ingl *continuous integration*) süsteeme, mis muudavad protsessi mugavaks ja kvaliteetseks. Selle abil saavad erinevad arendajad oma tööd koodihoidlaga katkematult mestida (ingl *merge*), saades automatiseeritud testidelt vahetut tagasisidet ning kiirendades sellega tööde kulgu.

1.2. Mittefunktsionaalne testimine

Funktsionaalse testimise kõrval on oluline roll mittefunktsionaalsel testimisel, kus ei keskenduta kindlatele funktsionaalsustele, vaid üldistele kvaliteediomadustele. Peamised sinna alla kuuluvad kategooriad on näiteks:

- jõudlus ehk süsteemi toimimine suure koormuse all;
- turvalisus ehk erinevate osade kaitstus rünnakute eest;
- kasutatavus ehk lihtsus süsteemi lõppkasutajale;
- ühilduvus ehk süsteemi toimimine erinevates keskkondades ja teiste komponentidega koostöös.

Mittefunktsionaalsed kvaliteedinõuded peaksid olema vähemalt sama olulised kui funktsionaalsed [7]. Jarzębowiczi ja Weichbrothi uuringu [8] kohaselt saavad need paraku pigem vähem tähelepanu ja ei ole ka nõuetes funktsionaalsete aspektidega võrdväärselt kajastatud. Tuuakse välja, et kliendid ja kasutajad pööravad tähelepanu rohkem sellele, mida süsteem tegema peaks ning seetõttu jäävad mittefunktsionaalsed nõuded pigem tahaplaanile. Samas määravad just need omadused suurel määral kasutajakogemuse ja rahulolu tootega.

Seetõttu on Camacho jt rõhutanud [7], et isegi kui klient ise ei pruugi neid aspekte esiplaanile seada, on siiski tähtis neile nõuetele äriline prioriteet määrata. Nad uurisid, mis täpsemalt mõjutab mittefunktsionaalsete nõuete rakendamist välearendust kasutavates meeskondades ning küsisid asjaga kokku puutunud spetsialistide arvamust. Nende intervjuudega tuli välja, et lisaks prioriteedile ja ajale on olulised ka teadlikkus ja teadmised, mis võivad funktsionaalsest poolest olla hoopis erinevad. Need nõuded võivad firmalt vajada spetsiaalsete inimeste palkamist, kogenenumate töötajate suuremat panust ning teiste projektide prioriteedi langetamist [7]. Mittefunktsionaalse testimise meetodid on omavahel erinevad ning järgnevalt on peamiseid kategooriaid lähemalt kirjeldatud.

1.2.1. Jõudluse testimine

Üks oluline tarkvara osa on tema jõudlus ehk võime suurema koormuse all vastu pidada. See tähendab seda, kas tarkvara toimetab teatud koormuse all piisavalt kiiresti, et kasutajakogemus ei kannataks. Kui näiteks süsteem peale arendust reageerib kasutaja sisendile tunduvalt kauem, siis kujutab see endast probleemi. Jõudlust võivad mõjutada nii suurenenud kasutajate arv kui ka andmehulkade tõusmine. Jõudlustestide sooritamise võimalused jagatakse mitmeks erinevaks meetodiks [9]:

- Koormustestimine (ingl *load testing*) aitab välja selgitada, kuidas süsteem mingile kindlale koormusele vastu peab. Üldjuhul üritatakse jäljendada reaalset olukorda kasutades süsteemi kindla numbri kasutajate poolt või tehakse teatud arv päringuid.
- Pingustestimise (ingl *stress testing*) korral tekitatakse koormus, mis on tavakasutusest suurem, et välja selgitada, mis on rakenduse võimekuse piir. See annab lisaks ka võimaluse jälgida muutusi, mida uued arendused tekitavad.
- Kestvustetimine (ingl *soak testing*) kontrollib rakenduse toimimist koormuse all pikema aja jooksul. Jälgitakse, kuidas rakendus vastu peab ja millisel määral tema käitumine muutub.
- Mastabeeritavuse testimise (ingl *scalability testing*) käigus vaadeldakse, kuidas süsteem käitub lühiajaliste järskude koormuste all, mis aitab näiteks jäljendada uute kliendigruppideni jõudmist.

Üldises plaanis peaks jõudluse testi juures eesmärk olema valmistusjärguga võimalikult sarnase või veelgi suurema koormusega seisundi tekitamine ning süsteemi kiiruse,

mastaapsuse ja stabiilsuse kontrollimine. Oluline on, et olemas oleks võrdlusmoment ehk testimise käigus oleks võimalik võrrelda tulemust varasema olukorraga [10]. Jõudluse hindamiseks ekstreemsetes oludes on võimalik kasutada mitmeid abivahendeid, populaarsemateks on Radview WebLOAD[1], LoadNinja[2], J Meter[3] ning Load Runner[4], mida saab Jenkinsi[5] või Seleniumi[6] abiga kasutada ka automatiseeritud testides ja pidevintegratsioonis [11].

1.2.2. Turvatestimine

Küberrünnakud on tänapäeva ühiskonnas kahjuks laialt levinud probleem ja on muutumas järjest aktuaalsemaks. Üksikud õnnestunud rünnakud võivad hävitada ettevõtte maine, avalikustada delikaatseid andmeid ja kasutajate usaldus võib igaveseks kaduda. Turvatestimise eesmärk on üles leida ohud, turvaaugud ja haavatavused, et ära hoida laiaulatuslik kahju.

Esimene variant turvalisust parandada on visuaalne koodi vaatlus. Lähtekoodi uurimine võib käia nii automatiseeritud tarkvara kui kogenud spetsialisti abil. Automaatne protsess on kiirem ja vajadusel suuremahulisem, kuid üldjuhul ei arvesta see nõuete ja äri eripäradega. Enim kasutatavate vahendite seas on näiteks Veracode[7], Checkmarx[8] ja Burp Suite Professional[9] [12]. Inimesepoolne kontroll võib olla spetsiifilisem, kuid nõuab rohkem ressursse ning probleemid võivad ka sellisel juhul märkamatuks jääda.

Teine turvalise kvaliteedi tagamise viis on tarkvara eetiline ründamine. See võib toimuda nii meeskonna siseste testijate kui ka eraldi palgatud ettevõtete kaudu. Tüüpilised vead, mida selle käigus avastada, on ebapiisav sisendi validatsioon, puhvri ületäitumine ning valesti konfigureeritud õigused ja turvaseaded [13]. Väga laialdase ja pidevalt muutuva olemuse tõttu on turvatestimine keskmisele kvaliteedi tagamise insenerile väljakutsuv ülesanne.

[1] https://www.radview.com/
[2] https://loadninja.com/
[3] https://jmeter.apache.org/
[4] https://www.microfocus.com/en-us/products/loadrunner-professional/overview
[5] https://www.jenkins.io/
[6] https://www.selenium.dev/
[7] https://www.veracode.com/
[8] https://checkmarx.com/
[9] https://portswigger.net/burp/pro

1.2.3. Kasutatavuse testimine

Kasutatavus väljendab, kui lihtne on rakenduse või veebilehega toimetada. Kasutajate mugavuse ja efektiivsuse hindamiseks on heaks vahendiks kasutatavuse testimine klientide peal. Neile antakse täitmiseks kindlad ülesanded, jälgitakse nende täitmist ja täheldatakse murekohti. Kasutatavuse juures on oluline, kui lihtne ja kiirelt kasutatav on süsteem esimesel kokkupuutel, kuidas see inimestele meelde jääb ning milliseid eksimusi nad teevad [14].
Sarnase lahendusena saab välja tuua ka beetatestimine, mille puhul osadele lõppkasutajatest tehakse uus versioon varem kättesaadavaks ning nende tagasiside põhjal leitakse üles võimalikud probleemid. Eesmärgiks on kindlus, et süsteem on tegelikkuses sihtide saavutamiseks piisavalt selge ja lihtne. Vajadusel saab teada kitsaskohad, mida enne toote lõplikku avalikustamist parandada. Lisaväärtusena aitab see paremini tundma õppida kasutajaid ja kliente.

1.2.4. Ühilduvuse testimine

Rakenduse kasutajaid ja nende eelistatud tarkvarasid on mitmesuguseid ning sellega võib kaasneda erinevaid probleeme. Ühilduvuse testimise eesmärk on vaadelda, kuidas rakendus erinevates keskkondades toimib ja oma ülesandeid täidab. See sisaldab näiteks erinevate seadmete, operatsioonisüsteemide, brauserite ja andmebaaside kasutamist. Selliste omavaheliste ühenduste kontrollimine võib kiiresti muutuda keeruliseks, kuna erinevaid kombinatsioone leidub väga palju [15]. Üks võimalus on toetada ainult kindlaid valikuid, kuid see eeldab klientide kasutuseelistuste tundmist või sellekohaseid märkmeid nõuetes. Teine võimalus aja säästmiseks on kasutada vastavaid abirakendusi või virtuaalmasinaid.
Kõik mainitud testimismeetodid on olulised ja erinevates situatsioonides vajalikud. Funktsionaalsed ja mittefunktsionaalsed nõuded vajavad kvaliteedi kontrollimisel mõnevõrra erinevaid teadmisi, kuid vaatamata sellele nõuab nende testimine igal juhul süsteemset panust. Seevastu on reaalne maailm tihtipeale teooriast küllaltki erinev ning on loogiline, et mõningaid lahendusi tuleb kohandada ettevõtete spetsiifikast ja vajadustest lähtuvalt.

2. Testimine Playtech Estonia osakonnas

Käesolevas peatükis antakse ülevaade töö metoodikast ning selgitatakse, kuidas toimub projekti erinevate osade kvaliteedi tagamine Playtech Estonia uuritavas osakonnas. Eraldi käsitletakse nii funktsionaalset kui ka mittefunktsionaalset testimist. Tulemused pärinevad iseseisval uuringul ning intervjuudest tuleneval informatsioonil. Viimases alapeatükis on eraldi välja toodud võimalused, kuidas tulevikus tööde kulgu parandada.

2.1. Töö metoodika

Töö praktiline osa põhineb Playtech Estonia ühe osakonna näitel. Playtech on maailma üks suurimaid mängutarkvara tootjaid, mis jõuab platvormi ja terviklahendusi pakkudes sadade miljonite mängijateni üle maailma [16]. Ettevõte on loodud Eestis ning tänase päevani on siinne arenduskeskus firma olulisimate seas. Playtechi üks tähtsaim äriüksus on Eestis paiknev IMS (*Information Management System*), mille kaudu saavad operaatorid täieliku ülevaate ja kontrolli mängijate tegevuste ning süsteemi toimimise üle. Tänu IMSi platvormile eristub ettevõte mitmetest oma konkurentidest. Antud alapeatükis selgitatakse uuritava osakonna struktuuri ning meetodeid, kuidas selles kvaliteedi tagamise protsesse uuriti.

2.1.1. Uuringu sihtmärk

IMSi hulka kuulub andmeteenuste ja raporteerimise (ingl *Data Services and Reporting*, edaspidi DSR) osakond, kus on kokku 33 töötajat. Struktuurselt jaguneb see kaheks üheksaliikmeliseks arendusmeeskonnaks ning üheks kuueliikmeliseks kvaliteedi tagamise üksuseks. Arendajad on jagunenud sündmuste platvormi (ingl *Event Platform*) ja raporteerimise teenuse (ingl *Reporting Services*) meeskonnaks ning igapäevatöös tegelevad nad erinevate toodetega. Testijate meeskond keskendub peamiselt käsitestimisele (ingl *manual testing*), kontrollides kvaliteeti mõlema arendusmeeskonna töös. Testijad teevad arendajatega tihedat koostööd ning lõpptulemusena kontrollivad nende tööde korrektsust. Peale arendajate ja testijate kuuluvad meeskonda erinevad juhid, analüütikud ning *DevOps* insenerid. Lisaks on osakonnal igapäevane seos spetsialiseeritud meeskonnaga, kes tegeleb

automatiseeritud testide loomise ja pideva jälgimisega. Töö DSRi osakonnas kulgeb välearenduse printsiipide alusel.

Playtech Estonia DSRi osakonnas on peamiseks ülesandeks andmete erineval kujul raporteerimine ja kuvamine regulaatoritele, kasutades sealhulgas firma enda loodud rakendusi ning platvorme. Seadustest ja regulatsioonidest tulenevalt on korrektsete ja õigeaegsete andmete edastamine ülioluline ning kõrvalekalded võivad viia lepingute lõpetamiste ja trahvideni ning tekitada seeläbi kogu ettevõttele probleeme. Eelneva tõttu on DSRi osakonnas kvaliteedi tagamine ja sellega seonduv väga tähtis ning sellele pannakse suurt rõhku.

2.1.2. Ettevalmistus intervjuudeks

Bakalaureusetöö käigus hindas autor kõigepealt süsteemi toimimist iseseisvalt ning proovis endale saada ülevaate, kuidas näeb testimisprotsess välja Playtech Estonia DSRi osakonnas. Ta tutvus nii arendajate, käsitsi testijate kui ka automatiseeritud testide loojate tööga. Meeskonnasiseselt on ettevõttes loodud materjale, mis sisaldavad kasutavate tarkvarade nimekirju ja lühiülevaateid. Need on kasulikud uutele töötajatele, kuid mõeldud ka kogenumatele testijatele informatsiooni ja teadmiste lihtsaks jagamiseks. Autor viis end kõikide leiduvate asjakohaste materjalide ja dokumentatsiooniga kurssi ning osales nii meeskonnasisestes kui planeerivatel koosolekutel. Vajadusel või täpsustavate küsimuste korral suhtles ta seotud üksuste liikmetega ning omandas ülevaate nende tööülesannetest ja vastutusaladest. Inimesed kirjeldasid, kuidas nad probleemidele lähenevad ning erinevate testimismeetodite puhul käituvad. Autor viis ennast kurssi ülesannete jaotusega ning sai ülevaate lõpptootesse jõudnud defektidest. Kõik eelnev andis aimduse, millele testimisel suuremat tähelepanu pööratakse ja millised on olulisemad nõuded.

2.1.3. Intervjuude metoodika

Iseseisvatele uuringutele lisaks viis autor 2022. aasta märtsi kuu jooksul läbi neli, keskmiselt 40 minutit kestnud, poolstruktureeritud intervjuud. Intervjueeritavateks valiti kolm Playtech Estonia uuritavas osakonnas juhtival positsioonil ning asjadega kursis olevat isikut – kvaliteedi tagamise meeskonna eestvedaja (J), sündmuste platvormi arendusüksuse pealik (T) ja raporteerimise teenuse juht (N). Lisaks juhtidele kuulus intervjueeritavate hulka üks kogemustega seenior - kvaliteedi tagamise insener (R). Valik osutus nende kasuks, kuna välja

toodud neli inimest on saanud testimisprotsesse aja jooksul enim soovitud suunas mõjutada ning arvamused peegeldavad testimise olemust kõige täpsemalt. Vastused ja ideed kujunesid valimil küllaltki sarnasteks, mistõttu ei olnud vajadust intervjueeritavate hulka suurendada. Küsimused (vt lisa 1) hõlmasid endas töö esimeses osas käsitletud teemasid. Intervjueeritavatele anti võimalus olukorda enda vaatest kirjeldada ning välja pakkuda ideid olukorra parandamiseks (vt lisa 2). Lisaks funktsionaalsele ja mittefunktsionaalsele testimisele puudutasid küsimused välearenduse meetodite mõjusid ning Playtechi spetsiifilisi teemasid. Koos intervjueeritavatega prooviti mõista, kas kõrvalekalded tavapärastest levinud töövõtetest on tahtlikud ning kas osakonnas saaks kasutusele võtta mingeid uuenduslikumaid lähenemisi.

2.2. Funktsionaalne testimine

DSRi osakonna projektis algavad uute funktsionaalsuste lisamine ja muudatuste tegemine projektijuhi otsustest ning kõigepealt teeb analüütik vajadusel analüüsi. Seejärel läheb tööjärg arendajatele, kes teevad nõutava valmis ning lisavad võimalusel juba selles faasis väiksemamahulised esimesed testid. Samuti jätavad nad käsitsi testijatele lühikese kirjelduse, mida arendaja arvates oleks võimalik testida.

2.2.1. Üksuste testimine

Esimene etapp, kus funktsionaalsused testidega kaetakse, on üksuste testimine. Selle eest vastutavad arendajad ning eesmärgiks on kontrollida, et tähtsamad meetodid ja klassid teevad loogiliselt seda, mida nad tegema peavad. Põhiprogrammi (ingl *back-end*) testid on kirjutatud JUnit[10] ning eessüsteemi (ingl *front-end*) testid Vue.js[11] komponenditestimise abiga. Kui arendajal on muudatused valmis ja meeskonnakaaslastelt heakskiit olemas, saab töö eelmise versiooniga mestida. Selle käigus kontrollib süsteem, et ükski varem lisatud test poleks katki läinud ning alles peale seda muutuvad muudatused erinevates keskkondades kättesaadavaks. See tähendab, et kui midagi on sellel tasandil katki läinud, tuleb arendajal see igal juhul korda teha või testid uue lahendusega vastavusse viia. Seega on tegemist tavalise pidevintegratsiooni süsteemiga ning intervjueeritavate sõnul toimib see hästi.

[10] https://junit.org/junit5/
[11] https://vuejs.org/

Üksuste testidega on kogu koodihoidlast sõltuvalt projektist kaetud 30 kuni 50 protsenti ridadest. See number on väiksem kui teoreetilise poole pealt soovituslik oli [5]. Ühes arenduse üksuses on olukord parem kui teises, kuid mõlemad meeskondade juhid nõustuvad, et protsent võiks olla suurem. Samas protsendi kunstlik suurendamine ei annaks positiivset efekti, nagu arvab intervjueeritav (T): "Olulisem on katta spetsiifilised äriloogikad ja näiteks iseloodud andmestruktuuride toimimine". Samuti rõhutavad mõlemad, et tähtsama osa moodustavad ettevõttes automatiseeritud integratsioonitestid. Seevastu rõhutati, et uutes projektides proovitakse üksuse ja komponenditestid tuua järjest olulisemale kohale. Praegune suund on kogu uus funktsionaalsus testida üksuse põhiselt ning kui see pole võimalik, siis kasutada selle asemel integratsiooniteste.

2.2.2. Käsitsi testimine

Sisuline funktsionaalne testimine on jäänud suuremalt jaolt testijate meeskonna ülesandeks. Üldjuhul on protsess selline, kus peale arendaja tegevusi paneb testijate meeskonna liige muudatustega versiooni testkeskkonda. Seejärel kontrollitakse vastavalt ülesande püstitusele, loodud analüüsile ja arendaja soovitustele, kas kõik toimib nii, nagu planeeritud oli. Tihtipeale on ka testijatel endil vaja täiendada dokumentatsiooni ning suhelda analüütikute ja arendajatega. Kui avastatakse nõuetest kõrvalekaldeid või defekte, siis registreeritakse vead ning lastakse arendajal need parandada. Testimisprotsess ise toimib üldiselt töötajate hinnangul probleemideta. Intervjuudest lähtuvalt on tulemustega rahul nii testijate kui arendajate juhid. Suuremaid probleeme pole valmistusjärku (ingl *production*) jõudnud ning seal esinenud muresid pole testkeskkonnas olnud üldjuhul võimalik üldse tekitadagi. Testija (R) intervjuust tulenevalt on süsteem hea, tal on pidepunktiks dokumentatsioon ja arendaja kommentaarid ning nendest lähtuvalt saab ta edukalt katta nii positiivseid kui ka negatiivseid juhtumeid.

Üksuste juhtide sõnul testitakse käsitsi läbi ligi 90 protsenti kõikidest arendustest ja intervjuude põhjal ei taheta seda protsenti ka allapoole tuua. Testijate meeskonna juht nõustub, et testimisel on kogu meeskonna edus väga suur roll ning käsitsi vaadatakse üle peaaegu kõik arendused. Senimaani on leitud vigade hulk ja paranduste vajadus olnud küllaltki suur ning enamikest suurematest arendustest on esile kerkinud mõned defektid. Ka mõlema arendusüksuse juhid nõustuvad, et kvaliteedi tagamisel on osakonnas väga oluline

osa. Üks nendest (T) kinnitas: “Käsitsi testimine annab samale teemale kõrvaltvaataja hinnangu ning kindlustab kõikidele osapooltele suurema turvatunde.” Lisaks on hetkel mõningates olukordades vaja manuaalselt üle kontrollida, et eelnevad funktsionaalsused toimima jäävad. Intervjueeritavad nõustuvad, et see on koht, kus saaks käsitsi tehtava töö mahtu vähendada ja kaasaegsemaid protsesse rakendada.

2.2.3. Välised tarkvarad

DSRi osakonnas on peaaegu kõikidele funktsionaalsustele loodud ka vastavad kasutajaliidesed (ingl *user interface*). See muudab mitmetes olukordades testimisprotsessi lihtsamaks, kuna veebirakenduse kaudu on võimalik sooritada lõppkasutajatega sarnaseid tegevusi. Samas testimiste käigus, eriti käsitsi sooritatava kontrolli puhul, kasutatakse vajadusel erinevaid väliseid tehnoloogiaid. Näiteks rakendusliidestele (ingl *application programming interface*) päringute saatmiseks kasutavad testijad lisaks SoapUId[12] ning Postmani[13]. Mõlema rakendusega on ilma suuremate eelteadmisteta võimalik sooritada REST-päringuid ja selle käigus luua spetsiifilisi testimise stsenaariume. Intervjuu põhjal eelistavad testijad võimalusel Postmani, kuna seal on veebiaadressi, saadetava sisu ja päringu tüübi määramine mugavam. Mõlema rakenduse puhul on aga probleemiks, et päringud pannakse kokku iga töötaja poolt eraldi. Kuigi neid saab vajadusel jagada, on see siiski ajaliselt lisakulu. Selle lahendusena on Playtechis lisaks loodud keskkond PIITS (*Playtech IMS Integration Tests Solution*), milles Java programmeerimiskeeles luuakse korduvkasutatavaid päringuid, mida sarnaselt koodi hoidlatega (ingl *repository*) on mugav kõikide osalistega versioonihaldussüsteemi Git[14] abil jagada. Intervjueeritud testija (R) nõustus, et kasutusel olevad tarkvarad teevad oma töö korralikult ära, kuid Playtechi siseselt, enda vajaduste põhjal loodud tarkvara on siiski kõige mugavam kasutada. Testijate meeskonna juht (J) on samal meelel ning kinnitab, et järjest enam kasutatakse enda loodud PIITS süsteemi, kuna seal on töötajatel võimalik toetuda teineteise teadmistele.

[12] https://www.soapui.org/
[13] https://www.postman.com/
[14] https://git-scm.com/

Rakendustest kasutatakse lisaks veel Puttyt[15] turvalise ssh ühenduse loomiseks ja erinevate andmebaaside tööriistu nagu PL/SQL[16], pgAdmin[17] ning MySQL Workbench[18]. Nende rakendustega on kõik töötajad intervjueeritavate hinnangul mõningate mööndustega rahul ning alternatiivide otsimise vajadust pole olnud.

2.2.4. Automatiseeritud testimine

Üksuse ja käsitestidele lisaks kontrollivad DSRi osakonnas süsteemi toimimist automatiseeritud testid, mis on jagunenud kahe erineva projekti vahel. Nende eesmärk on tagada, et varem töökorras olnud teenus poleks katki läinud nii integratsiooni kui kogu süsteemi põhiselt. Üheks keskkonnaks on PIITS, mida lisaks andmete genereerimisele kasutatakse ka automatiseeritud testimise juures. Selle haldajateks on arenduse ja testijate meeskonnad. Testide loomisesse panustavad mõlemad, olenevalt vajadustest ja oskustest. Need testid kontrollivad peamiselt rakendusliideste toimimist ning nende abil saavad kõik töötajad vajadusel kiiret tagasisidet. Testid käivituvad igaöiselt ning lisaks saavad neid kasutada nii arendajad tarkvara loomise käigus kui ka testijad kvaliteedi tagamise protsessi jooksul. PIITSi puhul on mugav, et kõik päringud, mis näiteks käsitsi testimise käigus tehakse, saab muuta edaspidisteks automatiseeritud testideks. Intervjuudest tuli vastavale platvormile ainult positiivseid hinnanguid. Kõik püsib ühes kohas ning tänu sellele annab kiirelt hea ülevaate. Platvormi kaudu saavad kvaliteeti panustada ühiselt nii testijad kui ka arendajad. Intervjuudest lähtuvalt saaks PIITSi positiivset mõju veelgi suurendada, kuna hetkel leidub veel funktsionaalsusi, mille toimimist kontrollitakse korduvalt käsitsi. Kõik need tegevused oleks tulevikus efektiivsuse tõstmiseks vajalik automatiseerida.

Teine täiesti eraldiseisev automaatselt jooksvate testide süsteem käivitub igal ööl eraldi selle jaoks mõeldud pidevintegratsiooni keskkondades. Neid teste loob eraldi meeskond, kes probleemide või avastatud vigade korral neid ka DSRi meeskonnale raporteerib. Kirjelduse, mida test peaks kontrollima, loovad DSRi testijad kui nad on enda töö lõpetanud. Küsimuste korral juhendavad nad automatiseeritud testidele keskendunud meeskonda testide loomisel ning uurivad põhjuseid, kui testid hiljem probleeme kajastavad. Nende testide oluline lisakasu seisneb selles, et need kinnitavad, et kõige tähtsamad funktsionaalsused on jäänud

[15] https://www.putty.org/
[16] https://www.oracle.com/database/technologies/appdev/plsql.html
[17] https://www.pgadmin.org/
[18] https://www.mysql.com/products/workbench/

tööle. Sellest tulenevalt on nõutud, et enne valmistusjärku paigutamist (ingl *deploy*) peavad olema need edukalt läbitud. Automatiseeritud testide meeskonnaga suhtlevad põhiliselt käsitsi testijad. Nende sõnul toimib töökäik hästi, kuna igal ööl jõuab testkeskkonda värskeim versioon ja järgmine hommik on nende kohta kõikide testide tulemused olemas. Probleemina toovad nad välja selle, et kõikide teenustega selline süsteem veel ei toimi ja versioonide paigutamisel on mõnel juhul veel liigselt automatiseerimata tööd.

2.2.5. Dokumentatsioon

Üldiselt on kõik neli välearenduse tunnust DSRi osakonnas kasutusel. Inimestevaheline suhtlus on tihe ja toimiv, klientidega suheldakse aktiivselt ja muudatused projektis võetakse positiivselt vastu. Sealhulgas on kvaliteedi tagamine kaasatud projekti algstaadiumist alates, kuna testijad osalevad planeerivatel koosolekutel ning on kaasatud erinevatesse aruteludesse. Samuti on töötav tarkvara dokumentatsioonist prioriteetsem, kuid viimast peavad intervjueeritavad siiski suures ettevõttes oluliseks. Playtechis on prioriteet alati defektideta toode, kuid dokumentatsioon peab suuremate ja keerukamate arendustega kaasas käima. Sisuhaldustarkvaras Confluence[19] jagavad vastavasisuliste dokumentidega teadmisi nii analüütikud, arendajad kui ka testijad. Mingites olukordades on see kindlasti vajalik, kuid tihtipeale leidub erinevaid situatsioone, kus dokumentatsiooni saaks vähendada või parandada. Leidub projekte, kus analüütik teeb tarkvaranõuete spetsifikatsiooni ning hiljem teeb sellega sarnase dokumendi ka testija. Nende fookus on erinevatel punktidel, kuid vaatamata sellele leidub kirjutistes mõningast ülekatet. Teine keerukus avaldub selles, et põhjalikku dokumentatsiooni on muutuva projekti juures raske hoida ajakohasena. Eelnevast tulenevalt jäävad nõuetena üldjuhul kajastamata mittefunktsionaalsed vajadused ja kasutajagruppide täpsed nõuded ja eripärad.

Kõikide intervjueeritud meeskondade juhid on ühel meelel, et dokumentatsiooni roll peaks olema veel palju suurem. Nende hinnangul oleks enamike toote osade kohta vajalik nii äriline, tehniline kui ka kasutajatele suunatud dokumentatsioon. Testijate juhi (J) sõnul oleks võimalik täpse analüüsi ja tootekirjeldusega kokku hoida nii arendajate kui ka testijate aega. Dokumendid peaksid olema alati asjakohased ning tema hinnangul ei tohiks ilma selge kirjelduseta toote osa testi jõuda. Samas teiste töötajate ja intervjuus osalenud testija (R)

[19] https://www.atlassian.com/software/confluence

hinnangul see nii suur probleem ei ole. Suurema murena toob ta välja kliendigruppide mitte tundmist. Seda saaks parandada, kui lisaks tehnilisemale kirjeldusele oleks kajastatud ka levinuimad kasutajalood, mida hetkel saab kogemuste ja loogika põhjal ainult aimata. Sama kinnitab ka arendusmeeskonna juht (T): "Oluline on nõuetest arusaamine, nende kaardistamine ja seejärel testimine."

2.3. Mittefunktsionaalne testimine

Kui teiste ettevõtete põhjal valminud analüüsid [7, 8] näitasid, et mittefunktsionaalsele poolele pööratakse vähe tähelepanu, siis sama kehtib üldiselt ka DSRi osakonna kohta. Mittefunktsionaalseid nõudeid ei peeta äriliselt nii olulisteks, kui seda on funktsionaalsed. "Mittefunktsionaalne testimine on kindlasti tähtis, kuid samas on põhifookus olnud alati peamiselt sellel, et funktsionaalse poole pealt toimiks kõik nõutud tasemel," tõdeb meeskonna juht (N). Mittefunktsionaalne pool on ka nõuetes kajastatud väiksemal määral ning kujuneb välja arendajate ja testijate kommunikatsioonis.

Kõige parem on olukord kasutatavuse testimise poolelt. Intervjueeritava (J) sõnul: "Isegi kui mõningates olukordades ei ole teada kliendi täpset eelistust, kujuneb analüütiku, arendaja ning testija suhtluses ja koostöös arusaadav toode". Lisaks saab tänu lühikesele kahenädalasele arendustsüklile ka kasutajatelt pidevalt tagasisidet. Siiski tunnistab testija, et kliendi vajadustest neil täielikku pilti ei ole. Kuna loodud rakendused on erinevad ning tarbijaid tavakasutajate, regulaatorite, litsentseerijate ning erinevate Playtechi enda töötajate näol on palju, siis on keeruline kõikide kasutuslugude peale mõelda. "Proovin ette kujutada, mis on kasutajate jaoks mugav ning arusaadav ja mida saaks parandada, kuid täpne ettekujutus nõuetest selles osas puudub," nentis testija (R).

Ühilduvuse testimise juures on oluline, et süsteemi erinevad komponendid omavahel ühendatult toimiksid. Kuna DSR osakonna lõikes on Playtechi äri suunatud teistele ettevõtetele ning mitte tavainimestele, siis on kasutajate hulk limiteeritud ning võimalik on teha teatuid lihtsustusi. See tähendab, et enamikes olukordades pole vajalik testida mitmeid integratsioone erinevate seadmete, operatsioonisüsteemide või brauserite vahel. Näiteks pole hetkel olnud oluline rakenduste toimimine telefoniekraanidel ning veebibrauseritest on toetatud ainult Google Chrome'i.

2.3.1. Jõudluse testimine

Jõudluse küsimus on DSRi osakonnas väga oluline ning ligi pooled valmistusjärgu defektidest on selle teemaga seotud. Andmebaasipäringud, millega andmeid raporteeritakse, peavad olema optimeeritud ja efektiivsed, kuna infot mängijate kohta erinevatel platvormidel on miljonites. Testimise meeskonna juht nõustub, et tihtipeale tekib olukord, kus testkeskkonnas ei ole võimalik sarnast andmete hulka imiteerida. Selliste olukordade jaoks on kasutusel süsteem, kus vastav raport laetakse raporti looja poolt üles lõppkasutajate keskkonda ning seejärel on võimalik päriselu andmestikul testida, kui kiiresti andmebaasipäringud suurtel mahtudel toimivad. Samas ei saa sellisel meetodil kontrollida, mis juhtub, kui andmete hulk tulevikus tunduvalt kasvab või struktuur muutub. Samuti puudub testimisel kindlustunne, kuna andmestik, millel päringuid jooksutatakse, on tundlik ning sealsed modifikatsioonid välistatud. Eelnev tähendab lisaks seda, et rakendada pole võimalik erinevaid jõudlustesti meetodeid ning abivahendina ei saa kasutada vastavaid rakendusi.

Sellel teemal rääkides elavnevad kõik intervjueeritavad. Testijad kinnitavad, et nemad seda osa ei kata ning kogu vastutus jääb arendaja kanda, kes peaks vajadusel oma tööd suuremate mahtudega testima. Lahenduseks võiks üksuste juhtide sõnul olla eraldi keskkond, kuhu tekitada massiivsed andmete ja päringute hulgad. See tähendaks, et kaoks ära vajadust viia testimist läbi valmistusjärgus, mis vähendaks kindlasti teatuid riske. Lisaks võiks aidata väliste tarkvarade rakendamine, mille abil oleks võimalik näiteks imiteerida suuremat veebirakenduste kasutajate hulka.

2.3.2. Turvatestimine

Turvatestimisele pööratakse testijate meeskonnas tähelepanu pigem harva. Tavalise arenduse käigus sellele pidevalt ei mõelda ning üldjuhul keskendutakse sellele vaid juhul, kui arendaja vastava muudatuse käigus millelegi viitab. Lisaks on protsessis kasutusel kaks turvalisusega seotud programmi. Esimene neist kasutab Checkmarxi[20] tarkvara, mis koodi ja kompositsiooni staatilise analüüsi käigus tuvastab probleemsed kohad. Teine sarnase

[20] https://checkmarx.com/

ülesehitusega projekt on olnud WhiteSource[21], mis keskendub peamiselt erinevate teekide asjakohasusele ja turvalisusele.

Arendajate juhtide sõnul on mõlemast olnud kasu, kuna aja jooksul on mõned väiksemad SQLi ja XMLi süstide (ingl *injection*) võimalused ja muud turvaaugud avastatud. Lisaks tõdevad töötajad, et osakonna spetsiifika tõttu pole turvatestimine senimaani väga oluline olnud, kuna süsteem on olnud kättesaadav ainult piiratud kogusele kasutajatele. Uute kasutajaliideste tulekuga olukord mõnevõrra muutub ning turvatestimine muutub järjest tähtsamaks. Playtechis töötavad ka eraldi spetsialistid, kes teatud aja tagant erinevate süsteemidega tutvuvad ning eetiliste rünnakutega võimalikke ohte tuvastavad.

Kokkuvõtlikult on turvatestimine Playtechis hästi korraldatud, üksikute ohtude avastamisel on parandused saabunud kiirelt ning valmistusjärgus pole selle teemalisi probleeme tekkinud. Intervjueeritavate sõnul on sellel teemal oluline keskenduda töötajate arengule ja teadlikkuse tõstmisele, millele aitavad kaasa järjepidevad koolitused. Testijate meeskonnas on antud valdkonnas toimunud ka jagunemine ning suurema huviga spetsialistid keskenduvad teemale rohkem.

2.4. Muudatusettepanekud efektiivsemaks kvaliteedi tagamiseks

Playtech Estonia DSRi osakonnas on testimisele pandud palju rõhku, kuid leidub olukordi, mida oleks võimalik paremini lahendada või alternatiivseid lahendusi kaaluda. Autor esitas bakalaureusetöö tulemusena neli põhilist ideed, mille parandamise järel võiks protsess muutuda tõhusamaks.

2.4.1. Dokumentatsioon

Esimene murekoht paikneb dokumenteerimises. Intervjueeritavate sõnul puudub osakonnas vajalik ressurss, et tooteid piisavas mahus kirjeldada. Täpsem nõuete ja toote kirjeldus abistaks testijatel oma tööd teha ja uutel inimestel kiiremini end meeskonna tööga kurssi viia, kuna osa informatsioon oleks kompaktsel ja hoomataval kujul olemas. Samuti vähendaks see väiksemate defektide hulka, mis on tekkinud ebakõladest kliendi vajaduste ja töötajate eelduste vahel. Näiteks ei osata ette näha mõningaid äärejuhte. Kuna mõne teema kohta tundub dokumentatsiooni olevat palju ning erinevate üksuste poolt loodud, siis esimese

[21] https://www.whitesourcesoftware.com/

lahendusena võiks ressursi suurendamiseks aidata lihtsalt tähtsamatele teemadele fokuseerimine ning vältida mitte vajaliku dokumenteerimist. Samuti on hetkel põhjalikult kirjas see, mida ja kuidas testiti, kuid efektiivsem oleks testitud sisu kajastada automatiseeritud testidega. Dokumentatsiooni vajalikkuse ja sisu teemal on ettevõttes planeeritud esimesed koosolekud ning nende põhjal võetakse vastu uued suunad ja esialgsed otsused.

2.4.2. Üksuste testimine

Teise arenguna tuleks kasuks üksuste testimise suurendamine. Suure tõenäosusega ei aitaks see oluliselt toote kvaliteeti tõsta, kuid kindlasti muudaks protsessi kiiremaks. Arendaja saaks oma koodi muudatuste kohta koheselt kvaliteetsemat tagasisidet ning säästaks mõningatel juhtudel testimisele kuluvat ressurssi. Lisaks annaks see arendajatele koodi toimimise kohta kergemini informatsiooni, kuna dokumentatsioon ei pruugi kõike hõlmata. Meeskondade juhid on soovitusega nõus ning üksuste testide osakaal on osakonnas järjest tõusmas.

2.4.3. Automatiseeritud testimine

Lisaks muutuks osakonnas testimise protsess tunduvalt lihtsamaks, kui regressioontestid oleksid täielikult automatiseeritud. See tooks endaga kaasa käsitsi tehtava töö hulga vähenemise ja suurema kindluse kvaliteedi üle uute toote versioonide puhul. Peamine selle puhul on kirjutada uutele funktsionaalsustele testid, et kui korra on teema manuaalselt üle vaadatud, saaks edaspidi kasutada automaatikat. Selles suunas liikumine võiks DSRi osakonnas minna küllaltki valutult, kuna PIITSi ja pidevintegratsiooni loogikaga saaks seda edukalt rakendada. Samas nõuab see toimiva süsteemi muutmist, kuna enamik testimistest on senimaani sooritatud käsitsi. Töötajad on plaanist teadlikud ning järjest enam salvestatakse tehtavaid tegevusi PIITSi keskkonda. Pikemas perspektiivis saaks testimist laiendada suunas, kus valmistusjärku jõuavad ainult arendused, millele on kirjutatud ka automatiseeritud testid.

2.4.4. Jõudluse testimine

Suurim probleem valmistusjärgus on olnud jõudlusega seotud teemadel. Murede leevendamiseks on vajadus tuua mittefunktsionaalset testimist rohkem igapäevastesse kvaliteedi tagamise süsteemidesse. Kuna jõudluse testimine on hetkel mastaapide erinevuste

tõttu praktiliselt võimatu, oleks vajalik testkeskkonda või eraldi selle jaoks loodud uuele platvormile suuremahuliste andmete tekitamine. Sellega nõustusid ka kõik intervjueeritavad ning kuna probleemi tõsidust mõisteti ka osakonnas laiemalt, siis on juba esimesed sammud lahenduste suunas tehtud. Osakonna siseselt on eesmärgiks käesoleval aastal luua jõudluse testide tarbeks eraldiseisev testkeskkond, kus oleksid andmehulgad vähemalt sama suured kui valmistusjärgus. Sellise lahenduse puhul liiguks vastutus testijate meeskonnale ning nad saaksid turvalises keskkonnas rakendada kõiki soovitud jõudluse testimise võtteid. Testid saavad vastavalt vajadusele olla nii automatiseeritud kui ka käsitsi käivitatavad. Selline keskkond võimaldaks tulevikus paremini integreerida ka erinevaid jõudluse testimise rakendusi.

Töö valmimise hetkeks (mai 2022) on kõikides punktides toimunud osakonnas esimesed arengud ning plaan tulevikus olukorda veelgi parandada on olemas.

Kokkuvõte

Käesoleva töö eesmärgiks oli anda ülevaade kvaliteedi tagamise protsessidest kaasaegses tarkvaraettevõttes. Põhilistest testimise meetoditest anti teoreetiline ülevaade, mille käigus tutvustati välearenduse printsiipe ning kirjeldati nii funktsionaalse testimise põhimõtteid kui ka mittefunktsionaalseid meetodeid. Seejärel uuriti, kuidas lähenetakse sarnastele teemadele Playtech Estonia DSRi osakonnas. Vaadeldi, kuidas on olukord lahendatud ning seletati lahti, millised süsteemi osad on kaetud automatiseeritud testidega ning milline roll on käsitestimisel. Uurimise ja töötajate intervjuude põhjal selgitati välja ettevõtte sisesed töövõtted ning töötajate suhtumine erinevatesse testimise meetoditesse.

Kvaliteedi tagamine toimib Playtechi uuritud osakonnas üldises plaanis hästi ning valmistusjärku jõuab defekte vähe. Rõhk on olnud funktsionaalsel testimisel ning senimaani on suur osakaal olnud käsitestimisel. Ettevõtte spetsiifika tõttu on saanud näiteks ühilduvuse ja turvatestimisele pöörata vähem tähelepanu, kuid vaatamata sellele nõuavad mittefunktsionaalsed omadused tulevikus prioriteetsemat lähenemist.

Töös esitati neli peamist parandusettepanekut edukamaks kvaliteedi tagamiseks. Soovitati parandada dokumentatsiooni, suurendada üksuste testimise ja automatiseeritud testide osakaalu ning pöörata jõudluse küsimustele suuremat tähelepanu. Esimesed sammud kõikide muudatusettepanekute parandamiseks on ettevõttes tehtud.

Uurimistöö jätkamiseks saaks sarnasel teemal uurida mõne teise tarkvaraettevõtte tegevust ja kvaliteedi tagamise protsesse. See annaks parema võrdlusmomendi ning oleks kergem ettevõtetele kogemuspõhiseid soovitusi jagada. Teiseks oleks hea uurida Playtech Estonia DSRi osakonna arengut mõne aasta pärast uuesti. Uurida saaks, milliseid muudatusi on ellu viidud ning kas tulemusena on kaasnenud positiivseid arenguid.

Viidatud kirjandus

[1] K. Beck, M. Beedle, A. Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries, J. Kern, B. Marick, R. C. Martin, S. Mellor, K. Schwaber, J. Sutherland, D. Thomas. Manifesto for Agile Software Development. 2001. https://moodle2019-20.ua.es/moodle/pluginfile.php/2213/mod_resource/content/2/agile-manifesto.pdf (22.03.2022)

[2] 15th Annual State Of Agile Report. *Digital.ai.* 2021. https://digital.ai/resource-center/analyst-reports/state-of-agile-report (22.03.2022)

[3] S. A. Deshpande, A. N. Deshpande, M. V. Marathe, G. V. Garje. Improving Software Quality with Agile Testing. *International Journal of Computer Applications,* 2010, vol 1, no. 22. https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.184.6024&rep=rep1&type=pdf (22.03.2022)

[4] J. Wegener, A. Baresel, M. Conrad, S. Sadeghipour. The interplay between Model Coverage and Code Coverage. *Conference On Computer Aided Systems Theory.* 2003. https://www.academia.edu/23481126/The_Interplay_between_Model_Coverage_and_Code_Coverage?auto=citations&from=cover_page (22.03.2022)

[5] C. Arguelles, M. Ivanković, A. Bender, Code Coverage Best Practices. *Testing Blog.* 2020. https://testing.googleblog.com/2020/08/code-coverage-best-practices.html (22.03.2022)

[6] E. Collins, A. Dias-Neto, V. F. d. Lucena Jr. Strategies for Agile Software Testing Automation: An Industrial Experience. *IEEE 36th Annual Computer Software and Applications Conference Workshops*, 2012, pp. 440-445. https://ieeexplore.ieee.org/document/6341616 (22.03.2022)

[7] C. R. Camacho, S. Marczak, D. S. Cruzes. Agile Team Members Perceptions on Non-functional Testing: Influencing Factors from an Empirical Study. *2016 11th International Conference on Availability, Reliability and Security*, 2016, p. 582-589. https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7784622 (22.03.2022)

[8] A. Jarzębowicz, P. Weichbroth. A Qualitative Study on Non-Functional

Requirements in Agile Software Development. *IEEE Access*, vol. 9, 2021, p. 40458-40475. https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9371679 (22.03.2022)

[9] R. Khan, A. Qahmash, M. Rashid Hussain. Soak Testing of Web Applications Based on Automatic Test Cases. *International Journal of Engineering Research and Technology. ISSN 0974-3154*, vol. 13, Number 12, 2020, p. 4746-4750. http://www.ripublication.com/irph/ijert20/ijertv13n12_95.pdf (22.03.2022)

[10] Devopedia. *Software Performance Testing.* https://devopedia.org/software-performance-testing (05.12.2021)

[11] N. Srivastava, U. Kumar, P. Singh. Software and Performance Testing Tools. *Journal of Informatics Electrical and Electronics Engineering*, Vol. 02, Iss. 01, S. No. 001, 2021, pp. 1-12. https://a2zjournals.com/jieee/uploadpdf/Nishi_1608186748P572.pdf (22.03.2022)

[12] Application Security Testing (AST) Reviews and Ratings. https://www.gartner.com/reviews/market/application-security-testing (05.12.2021)

[13] K. Scarfone, M. Souppaya, A. Cody, A. Orebaugh. Technical Guide to Information Security Testing and Assessment. *National Institute of Standards and Technology*. Special Publication 800-115. https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-115.pdf (22.03.2022)

[14] Usability 101: Introduction to Usability. https://www.nngroup.com/articles/usability-101-introduction-to-usability/ (05.12.2021)

[15] I. Yoon, A. Sussman, A. Memon, A. Porter. Effective and scalable software compatibility testing. *Proceedings of the 2008 international symposium on Software testing and analysis (ISSTA '08)*. p. 63–74. https://doi.org/10.1145/1390630.1390640 (22.03.2022)

[16] Meist - Playtech. https://www.playtech.ee/meist (22.03.2022)

Lisad

I. Intervjuude küsimused

1. Kui tähtis on testimine DSRi osakonnas? Mitu protsenti uutest arendustest umbes käsitsi testitakse? Kas tulevikus oleks võimalik hakkama saada ilma testijate meeskonnata või selle rolli osakonnas vähendada?
2. Millised omadused on testija puhul tähtsad? Kas pigem tehniline taip ja loogiline mõtlemine või teadmised testimise meetoditest?
3. Kui palju ühel arendusel keskmiselt testimise käigus defekte esineb? Kuidas seda vähendada?
4. Millest tulenevad valmistusjärgus tekkivad puudused ja defektid? Kuidas need sinna jõuavad? Milline on tagasiside testijatele kui tema kontrollitust need defektid läbi lähevad? Kas ja mida saaks teha paremini, et äärejuhte avastada?
5. Kas välearenduslikud lähenemised on midagi tööprotsessis muutnud? Kas nii palju dokumentatsiooni on vaja? Kas see on pidevalt asjakohane ja uuendatud?
6. Kui hästi testija tegelikult klienti tunneb ja nõudeid teab?
7. Miks loovad osa automatiseeritud teste meeskonna välised inimesed?
8. Kas mingeid protsesse võiks/saaks rohkem automatiseerida?
9. Kui suur osa toodetest on umbes automatiseeritud testide poolt kaetud? Kas midagi saaks selles osas paremaks muuta?
10. Välistest tarkvaradest kasutate näiteks SaopUId ja Postmani? Kuidas selline valik on kujunenud? Kas olete rahul või esineb mingeid suuremaid puudusi?
11. Mis võiks olla koodi katvus üksuse testide poolt? Kas selles osas annaks midagi parandada?
12. Kui levinud on testimisel põhinev arendus (ingl *test driven development*)? Kas seda saaks meeskonnas rohkem juurutada?
13. Kas nõustute, et mittefunktsionaalne testimine on seni jäänud pigem tagaplaanile?
14. Milline näeb hetkel välja jõudluse testimine? Kas ja milliseid jõudlustesti meetodeid neljast kasutatakse? Kas jõudlustestideks kasutatakse mingeid spetsiifilisi rakendusi?
15. Kas teil on teadmine, mis juhtub kui valmistusjärgu mahud tunduvalt kasvavad?

16. Mis võiks olla lahendus, et jõudluse muresid saaks avastada enne valmistusjärku? Kas suuremahuliste andmetega eraldi keskkond võiks aidata?
17. Kuidas käib hetkel turvatestimine? Kas turvatestimiseks kasutatakse mingeid spetsiifilisi rakendusi? Kas selle jaoks võiks olla meeskonnas eraldi inimesed või siis spetsialistid kogu ettevõtte peale või on see sarnaselt juba lahendatud?
18. Kuidas toimub kasutatavuse testimine? Kas kasutate mingil kujul beta testimist?
19. Milline näeb välja ühildatavuse testimine? Kui suure hulga tarkvaradega peab süsteem olema toimiv?
20. Millised on tulevikuplaanid testimise kontekstis? Mida saaks näiteks uutes projektides paremini teha?
21. Kas on testimisprotsesside kohta lisaks veel mingeid mõtteid?

II. Intervjuude vastused

Intervjueeritav J

Kui tähtis on testimine DSR osakonnas? Mitu protsenti uutest arendustest umbes käsitsi testitakse? Kas tulevikus oleks võimalik hakkama saada ilma testijate meeskonnata või selle rolli osakonnas vähendada?

Manuaalselt kaetakse kõik taskid, mis võimalik. Gruumingul arutatakse, kas on võimalik testida ning kui on, siis testitakse. Kui raporti inseneride pileteid mitte arvestada, siis kokku kaetakse manuaalselt testimistega umbes 85 protsenti jiradest. QA tiimijuhina, ütlen, et vähemaga ei saaks. Pole üldse seda meelt, et QA tiime poleks vaja, kuigi tean et see on populaarne teema. QA tiimid peaks ka tulevikus kindlasti olemas olema.

Millest tulenevad valmistusjärgus tekkivad puudused ja defektid? Kuidas need sinna jõuavad? Milline on tagasiside testijatele kui tema kontrollitust need defektid läbi lähevad? Kas ja mida saaks teha paremini, et äärejuhte avastada?

Production defektidest suur osa tuleb domeenist mis pole meie vastutada - statistika tabelid. Ei parandata meie poolt. Ülejäänud defektid tulevad sellest, et tava testkeskkonnas ei ole võimalik manuaalsel testijal neid testida, eelkõige jõudluse pärast, seda tahaks kindlasti parandada. Või siis äärejuhud, mida manuaalsel QA ei õnnestu testida. Eelmise aasta production defektid, kokku 265, neist 28 statistika, 10 performance, ülejäänutest enamik report viewer 3.0. äärejuhud, mis hetkel teadlikult parandamata ja see et pole jõudluse testide võimekust. Kui funktsionaalsust on halvasti testitud, siis räägitakse testijaga eraldi. Senimaani enamik juhtumeid on olnud sellised, et testkeskkonnas ei ole võimalik testida ja selle peale tulla.

Kas välearenduslikud lähenemised on midagi tööprotsessis muutnud? Kas nii palju dokumentatsiooni on vaja? Kas see on pidevalt asjakohane ja uuendatud?

Vanasti oli analüüs, aeg STPle ehk testiplaanile, mille kirjutamine võis käia paralleelselt arendusega. STP kirjutaja teine kui testija. Dokumentatsiooni võiks olla minu arvates kõvasti rohkem. Analüüs võiks olla täpsem, ühele arendustiimile oleks kindlasti analüütikut juurde vaja. Hetkeolukord kehva, testimine tugineb arendaja juhistel, mis ei ole tihtipeale piisav. Readme fail juba hästi. Kuna pole ressurssi DL analüütiku jaoks, siis dokumentatsioon on

arendajast lähtuv, mis jääb QA jaoks tihtipeale liiga keeruliseks. Dokumendid ilusti tegelikult asjakohased. Et oleks up-to-date võiks proovida siduda readme-ga, et iga commitiga hoida ka see värskendatud. Aga suures plaanis pole vahet kus, peamine et oleks olemas. Samas usability poole pealt, isegi kui mõningates olukordades ei ole teada kliendi täpset eelistust, kujuneb analüütiku, arendaja ning testija suhtluses ja koostöös minu arvates arusaadav toode.

Kas mingeid protsesse võiks/saaks rohkem automatiseerida? Kui suur osa toodetest on umbes automatiseeritud testide poolt kaetud? Kas midagi saaks selles osas paremaks muuta?

Kui inimesel on oskused ja soov, siis saab kirjutada ka autoteste. Mul poleks selle vastu midagi kui oleks spetsiaalne inimene, kes kirjutaks. Aga hetkel ei oleks sellel inimesel nii palju asju mida kirjutada. CI autotesti süsteem on minu jaoks hea. Toimib nii, et CIsse jõuab see latest versioon, iga õhtu, ehk enne südaööd võtab kõige värskemad komponendid, teeb deploy ja järgmine hommik on latestimate versioonide kohta kõikide testide tulemused. Kui saaks kõikide toodetega sama, siis oleks lähedal ideaalile.

Välistest tarkvaradest kasutate näiteks SaopUId ja Postmani? Kuidas need on kujunenud? Kas olete rahul või esineb mingeid suuremaid puudusi?

Senimaani kõik toimib. Enne SoapUId oli network mock API. SoapUId kasutati pikalt kuni saadi aru et midagi on puudu, puudu oli see et kuidas saaks osakondade vahel neid projekte jagada ilma et käid ja lähed copy pasted avastades et midagi on puudu. See oli toetatud SoapUI pro versiooniga, aga merge conflictid ja arutelud olid kohutavad. Rahaline väljaminek ei tasunud ennast ära. Kasvame Piitsa peale ja muud ei oskagi tahta. Kui oleks eraldiseisvaid tarkvarasid, mis teeks UIX kaudu perf testid kasutajatele lihtsaks, või security scripte, kas on midagi sellist oleks. Teed frontendist päringu ja hakkab ise proovima penetration teste näiteks.

Kas nõustute, et mittefunktsionaalne testimine on seni jäänud pigem tagaplaanil? Milline näeb hetkel välja jõudluse testimine? Mis võiks olla lahendus, et jõudluse muresid saaks avastada enne valmistusjärku? Kas suuremahuliste andmetega eraldi keskkond võiks aidata?

Seda põhimõtteliselt pole ning see on suur probleem. Performance testikeskkonna loomine erinevatele toodetele võiks olla eesmärk tõesti. Peab arutlema, kuidas saaks seda kasutada selliselt, et kõik konfiguratsioonid jääksid alles ja saaks igaöiselt jooksutada, mis oleks kõvasti parem. Läbi ajaloo on käinud kõik nõudepõhiselt. Kui testid ära jooksnud ja info

käes, siis kustutatakse keskkonnast konfiguratsioonid ära. Eraldi keskkonnad oleks head, mille järel oleks eesmärk katta seda QA poolt. Andmebaasi mahtude tekitamine ei ole probleem. Perf keskkonnad ei ole probleem tekitada mitmekordset prodi keskkonda. Raporti perf testi vastutusala on raport loojal. Sanity alla kuulub perf test vastu prodi. Tulevikus võiks olla selle jaoks ka eraldi keskkond.

Kuidas käib hetkel turvatestimine? Kas turvatestimiseks kasutatakse mingeid spetsiifilisi rakendusi? Kas selle jaoks võiks olla meeskonnas eraldi inimesed või siis spetsialistid kogu ettevõtte peale või on see sarnaselt juba lahendatud?

Teistel osakondadel rohkem turvatestimise teadmisi, kuna nende domeenil seda rohkem vaja. Lisaks on selle jaoks eraldi tiim, whitesource project mis käib UIsid läbi. Eraldi inimesed kes seda teevad. Playtechi sisene turvalisus, millega kogu kood käiakse nende loodud turvatestimise tööriistadega läbi. Eraldi inimest pole vaja, aga teadmisi tahaks tiimi sisse tuua. Praegu üks tiimiliige keskendub sellele ka rohkem. Sarnaselt autotestide kirjutajale, et oleks super kui keegi oskab seda, aga positsiooni pole pakkuda.

Milline näeb välja ühildatavuse testimine? Kui suure hulga tarkvaradega peab süsteem olema toimiv?

Playtech on öelnud, et peame toetama kõige levinumaid platvorme. Kuigi Google otsingu põhjal kõige levinum platvorm on Firefoxi, siis IMSi sees vähemalt on kokkulepe et testime Chromega ning kui töötab siis korras. Firefox mured lähevad kirja, kuid parandatakse juhul, kui tekib selleks vahendeid või ressursse. Vanasti ühe nurga peal oli mac, kes tahtis võis mingeid asju proovida.

Intervjueeritav T

Kui tähtis on testimine DSR osakonnas? Kas tulevikus oleks võimalik hakkama saada ilma testijate meeskonnata või selle rolli osakonnas vähendada?

Manuaalne QA annab samale teemale kõrvaltvaataja hinnangu ning kindlustab kõikidele osapooltele suurema turvatunde. Kui oleks kõik automaatne, siis midagi võib jääda kahe silma vahele, kuigi automaattestid lähevad läbi. See annab mulle ja ilmselt ka teistele parema turvatunde, et keegi on veel vaadanud otsa sellele arendusele, et see teeb seda, mis kirja sai.

Vähendamise koha pealt - kui me saaksime sellise lahenduse, et mingi uue arendusega tehakse manuaalse testimise käigus automaatne test. Siis järgmise muudatusega see test

jookseb automaatselt, et saaks keskenduda uutele asjadele. Kõik mis võimalik võiks katta ka automaattestidega. Hiljem tahame et see sama funktsionaalsus töötaks, kui seda ei muudeta.

Millest tulenevad valmistusjärgus tekkivad puudused ja defektid? Kuidas need sinna jõuavad?

Meie meeskonna peamised defektid tulevad sellest et raporti sisendandmed ei vasta sellele, mis on kokku lepitud või ei olnud kokkulepet ja tehti nii kuidas arvati, et on hästi, kuid tegelikult on vaja rakendada teisi tingimusi, mis pole piisavad või on liiga nõudlikud ehk ei vasta sellele, mis vaja on, näiteks, et number tohib olla ka negatiivne, aga lastakse läbi ainult positiivseid. Teine asi on jõudluse küsimused, kus andmete ja raportite hulk, mis juhtuvad on suured, siis neid testkeskkonnas tekitada on äärmiselt keeruline. Sellist situatsiooni on raske ette näha või isegi tagantjärgi keeruline uuesti tekitada ehk ka keeruline kontrollida et uuesti ei juhtu kui parandatud.

Kas välearenduslikud lähenemised on midagi tööprotsessis muutnud? Kas nii palju dokumentatsiooni on vaja? Kas see on pidevalt asjakohane ja uuendatud?

Teadvustatud probleem, et dokumentatsiooni on vähe. Pole analüüsi jaoks head süsteemi, pole toodete dokumentatsiooni. Seega pole ka testijal head ettekujutust. Testija võiks olla juba analüüsi dokumendi loomisel ning on üks valideerija, et ta oskab selle põhjal testida. Avaldub ka uute töötajate tulekul, uus testija või analüütik, arendajaga natuke teistmoodi.

Kui hästi testija tegelikult klienti tunneb ja nõudeid teab?

Nõuete esitamine analüüsile. Tegevused mida teeb näiteks devops, kas me need testime läbi testkeskkondades. Nendele pole üldjuhul UI autoteste. Kui nad ise väga oma vajadusi ei nõua, siis QA ei tea, mida selles osas tuleks testida. Ehk oluline on, nii devopsi tegevuste puhul kui ka üldiselt, nõuetest arusaamine, nende kaardistamine ja seejärel testimine.

Mis võiks olla koodi katvus üksuse testide poolt? Kas selles osas annaks midagi parandada?

See on erinevate toodetega erineval tasemel. Unit testidel peab olema mingi loogika, kui on lihtne toode, siis pole see nii oluline. Näiteks huvitab see, et andmed lähevad otsast sisse ja lõpuks on baasis. PIITS testist hetkel piisab, ei näe et oleks vaja suuri ümberkorraldusi. Kui tulevad sisse keerukamad loogika kohad, kus on vaja kontrollida kas loogika ja andmestruktuurid töötavad nii nagu me eeldame ja kui pole triviaalsed, siis nendele tuleks kindlasti testid teha. Ehk olulisem on katta spetsiifilised äriloogikad ja näiteks iseloodud andmestruktuuride toimimine. Sama põhimõtet oleme ka rakendanud, näiteks eraldi loodud

andmestruktuuriga, mis hoiab viimast 100t kirjet, sisemine struktuur, siis sellel korralikud testid olemas, et muudatused midagi katki ei teeks.

Kui levinud on testimisel põhinev arendus (ingl *test driven development*)? Kas seda saaks meeskonnas rohkem juurutada?

Seda pole, nõuab teistsugust mõtlemist ehk ei mõtle lahendusele vaid testile.

Milline näeb hetkel välja jõudluse testimine? Mis võiks olla lahendus, et jõudluse muresid saaks avastada enne valmistusjärku? Kas suuremahuliste andmetega eraldi keskkond võiks aidata?

Peaks vaatama, et ei tee seda esmases käsitsi testimise keskkonnas. Peab olema eraldi performance keskkonnas, võiksid olla automatiseeritud, et need jooksevad alati või käivitatakse PIITSi pealt. Toimuvad tegevused, genereeritakse datat. Samasse keskkonda ei saa panna, kuna seal tegeled sa mingite vigade tuvastamisega, kuna miljoni evendiga logi on juba keeruline jälgida, sellel kes manuaalselt üritab testida.

Kuidas käib hetkel turvatestimine? Kas turvatestimiseks kasutatakse mingeid spetsiifilisi rakendusi? Kas selle jaoks võiks olla meeskonnas eraldi inimesed või siis spetsialistid kogu ettevõtte peale või on see sarnaselt juba lahendatud?

Kui me hakkame välja paistma frontend rakenduste kaudu, siis muutume lõppkasutajate kaudu haavatavaks. Licenceed saavad teha pahasti nii tahtlikult kui tahtmatult. Siis turvaaspektid olulised. Siin on meil olemas turvalisusega tegelev meeskond. Soovitaks ka käsitsi testijatel saada sealt koolitusi, mida ja kuidas vaadata, et nad oskaks lihtsamad asjad üles leida. Lisaks Checkmarcks jooksutatakse mingi aja tagant. Edaspidi on töökorraldus selline, et nad jätkuvalt jooksutavad neid builde, kui nad midagi leiavad, siis teevad pileti. On leidnud sql injectioneid, xml injectioneid, vananenud librareid.

Milline näeb välja ühildatavuse testimine?

Kokkulepitud brauser, millega peab töötama - Chrome.

Intervjueeritav N

Kui tähtis on testimine DSR osakonnas? Mitu protsenti uutest arendustest umbes käsitsi testitakse? Kas tulevikus oleks võimalik hakkama saada ilma testijate meeskonnata või selle rolli osakonnas vähendada?

Alguses oli ainult manuaalne testimine. Kuna kõigil head ja halvad päevad, inimfaktor jääb ja erineva kogemusega töötajad, siis selle tõttu inimest ei saa asendada. Aga oleks vaja organiseerida automaattestimise protsessi, mis teeb lisavalidatsiooni, mida inimene ei saa korduvalt kontrollida. Lisaks arendaja teeb oma vaatenurgast, aga võivad teisi osi mõjutada, seda peaks katma autotest. Manuaalne osa lisakontoll ka sellele, mida arendaja on unustanud katta.

Kas välearenduslikud lähenemised on midagi tööprotsessis muutnud? Kas nii palju dokumentatsiooni on vaja? Kas see on pidevalt asjakohane ja uuendatud?

Olukord on parem kui osakonna teises tiimis, aga võiks kindlasti veel parem olla. Viimasel ajal on saanud suuremat fookust ning jah, proovime sellel teemal arutleda. Võiks olla äri dokumentatsioon inimkeeles, funktsionaalsused ja tehniline dokumentatsioon arenduse nüanssidega - andmebaasitabelid, JSON blokid. Kuidas süsteemid ja komponendi töötavad ja seotud on. Avalikele APIdele dokumentatsioon. Väikese tootega pole nii oluline, kuid suure tiimiga vaja kirjeldust hilisemaks. Kui kasutajad küsivad abi, siis see on spetsialisti aeg. Korraliku dokumentatsiooni korral ei pea seletama vaid saab viidata. Kasulik ka uuele inimesele ülevaade saamiseks.

Kas mingeid protsesse võiks/saaks rohkem automatiseerida?

Uute toodete puhul oleme tunduvalt rohkem kasutanud unitteste ning lisaks VUE komponenttestid, mis mõlemad aitavad, et buildimise käigus saaks juba teada. Lisaks kasutame aktiivselt PIITS platvormi. Uuel süsteemil osa VUE komponenttestid ja osa PIITS testid. Ja lisaks tundlikele funktsionaalsustele ja integratsioonidele eraldi autotestid. Üldiselt siis kolme tüüpi teste, autotestimist kindlasti vaja, annab kindlust. Teab juba arenduse käigus. Üldiselt autotest kindlasti aidanud. Vue ja PIITSi testid mõeldud rohkem arendajatele. Eraldi automatiseeritud testid mõeldud reliisimise mõttes, õige koht et muudatused ohutud ja toode töötab. CI olemasoleva RV jaoks töötab hästi.

Mis võiks olla koodi katvus üksuse testide poolt? Kas selles osas annaks midagi parandada?

Kui võimalik siis komponenttest, kui mitte siis PIITS test. Praeguses arenduses iga asja jaoks VUE komponenttest. Vanades projektides olukord natukene halvem.

Kas nõustute, et mittefunktsionaalne testimine on seni jäänud pigem tagaplaanil? Milline näeb hetkel välja jõudluse testimine? Kas teil on teadmine, mis juhtub kui valmistusjärgu mahud tunduvalt kasvavad? Mis võiks olla lahendus, et jõudluse

muresid saaks avastada enne valmistusjärku? Kas suuremahuliste andmetega eraldi keskkond võiks aidata?

Mittefunktsionaalne testimine on kindlasti tähtis, kuid samas on põhifookus olnud alati peamiselt sellel, et funktsionaalse poole pealt toimiks kõik nõutud tasemel. Samas ilma mittefunktsionaalse osata ei saa. Performance testi osas report vieweriga hea katvus nii manuaalse kui ka autotestiga. Kontroll kui kaua aega võtab, tuleb tagasiside arendajatele kui läheb liiga suureks. Raportitega oli mõte jõudluse teste kasutama hakata. Põhiraportid piitsa, et olla kindel et peale muudatust jääb toimima. Praegu katvust pole, plaan on. Testkeskkondades pole võimalik testida, ainuke koht production. Kuid ka seal on näiteks andmed väga erinevad - ühes vähe mängu tüüpe, aga väga intensiivne mängimine - statistika kompaktne, teises erinevad mängutüübid kasutusel. Perf testi tehes vastavat raporti tüübile valime keskkonna. Põhilised vead tulevad productionist - on automaatne süsteem, mis avastab probleemid. Eraldi keskkonna osas olen skeptiline, mahtude probleem. Genereeritakse andmed raporti jaoks, aga erinevaid võimalusi on palju (palju mängijaid, vähe tegevust. Teises vähe mängijaid ja palju tegevust). Suure süsteemi puhul keeruline. Aga olen nõus, et peaks alguses valima midagi ja siis sellest lähtuvalt edasi liikuma.

Kuidas käib hetkel turvatestimine? Kas turvatestimiseks kasutatakse mingeid spetsiifilisi rakendusi? Kas selle jaoks võiks olla meeskonnas eraldi inimesed või siis spetsialistid kogu ettevõtte peale või on see sarnaselt juba lahendatud?

2 projekti, Checkmarxsiga leiti mõned turvaaugud. Annab tead kui midagi valesti teeme. Teine Whitesource. Seal vaadatakse põhiliselt librareid. Lisaks on tehtud eraldi penetration teste, osaliselt manuuaalne ja osaliselt automaatne. Test eelmise aasta alguses, leiti kohad mida parandada - 40 lehekülge teksti soovituste ja piltidega, kuigi muresid oli vähe. See aasta uuesti RV3.0ga. Ehk päris mitu projekti ning kui turvaaugud leitud, siis kiiresti parandame.

Intervjueeritav R

Kui tähtis on testimine DSRi osakonnas? Mitu protsenti uutest arendustest umbes käsitsi testitakse? Kas tulevikus oleks võimalik hakkama saada ilma testijate meeskonnata või selle rolli osakonnas vähendada?

Hetkel väga tähtis, kuna tuleb ikkagi palju defekte välja. Automaattestimist võiks kindlasti suurendada ja manuaalset poolt vähendada. Playtechi suuruses ettevõttes on aga minu hinnangul mingeid inimesi ikka vaja.

Millised omadused on testija puhul tähtsad? Kas pigem tehniline taip ja loogiline mõtlemine või teadmised testimise meetoditest?

Tehniline taip on muidugi tähtis. Käsurida ja Postman pole hullu. SQL mitteteadmine teeb keeruliseks. Oluline on suhtlus, avatus ja julgus probleemidest rääkida.

Kas nii palju dokumentatsiooni on vaja? Kas see on pidevalt asjakohane ja uuendatud?

Oleneb komponendist. Uue regulatsiooniga olukord hea - raportid ja andmete liikumised. Tehniliste taskidega pigem kesisem.

Kui hästi testija tegelikult klienti tunneb ja nõudeid teab?

10 palli skaalal ma ütleks 5. Palju erinevaid tooteid, osad on licenciile mõeldud, osa regulaatorile, tech suport, devops. Proovin ette kujutada, mis on kasutajate jaoks mugav ning arusaadav ja mida saaks parandada, kuid täpne ettekujutus nõuetest selles osas puudub.

Kuidas näeb testimine välja? Kas arendaja suunised on mingis mõttes piiravad?

Võtan neid positiivse case-ina ja ülejäänu mõtlen ise sinna juurde.

Kas mingeid protsesse võiks/saaks rohkem automatiseerida?

Peaks olema suund, et kõike autotestida.

Kas ei oleks parem kui automatiseeritud testide tulemused oleksid olemas kohe peale uue versiooni paigaldust?

Kui oleks feature, et kohe siis oleks hea, aga ei häiri et tuleb järgmine päev. Sobib et algul saab ise näppida ja tutvuda ja siis tuleb autotesti tagasiside.

Kas ei oleks lihtsam neid teste enda meeskonna sees luua?

Oleneb kuidas seda korraldada. Kui peab meie tiimi inimesele seletama teemat, siis poleks vahet. Kui tema töö on ise varakult selgeks teha asi, siis ilmselt lihtsustaks. Pigem oleks parem aga praegune olukord ei häiri.

Välistest tarkvaradest kasutate näiteks SaopUId ja Postmani? Kuidas need on kujunenud? Kas olete rahul või esineb mingeid suuremaid puudusi?

Üldiselt rahul, pgadmin postgre kasutamiseks veidi tüütu, Netbeans ja muud andmebaasi UId tunduvad paremad, aga on tasulised. Hea meel et Piitsa datagen flowd tulid, oli tüütu et uue regulatsiooniga tuleb flowd teha, keerulisemate nõuetega vigade järgi confimine on tüütu. Postmanis saab võimalik, et liiga customizeda, aga samas saab teha spetsiifilisi päringuid.

Kas nõustute, et mittefunktsionaalne testimine on seni jäänud pigem tagaplaanil? Milline näeb hetkel välja jõudluse testimine? Kas ja milliseid jõudlustesti meetodeid neljast kasutatakse? Mis võiks olla lahendus, et jõudluse muresid saaks avastada enne valmistusjärku? Kas suuremahuliste andmetega eraldi keskkond võiks aidata?

Senimaani on production testinud. Oleneb taskist, aga pigem QA ei testi üldse. Proovin, aga tulemus ei anna enesekindlust, et tegelikult ka nii performib. Pigem vaatan lihtsalt et äkki on efekt juba väikese mahu peale. Eraldi keskkond kõlab hästi.

Kuidas käib hetkel turvatestimine? Kas turvatestimiseks kasutatakse mingeid spetsiifilisi rakendusi? Kas selle jaoks võiks olla meeskonnas eraldi inimesed või siis spetsialistid kogu ettevõtte peale või on see sarnaselt juba lahendatud?

Üldjuhul testime siis kui on teada, et arendus võib midagi sellist mõjutada või asi ise ongi turvalisusega seotud arendus. Isiklikult raske selle valdkonnaga kursis püsida. Eeldab et inimene ise huvitub ja on valdkonnaga kursis. Oleks hea kui näiteks üks inimene tiimist teab rohkem.

Milline näeb välja ühildatavuse testimine? Kui suure hulga tarkvaradega peab süsteem olema toimiv?

Toetame seda mis on kõige populaarsem. Kui muutub siis pole hullu läbi klõpsata ka mujal.

III. Litsents

Lihtlitsents lõputöö reprodutseerimiseks ja üldsusele kättesaadavaks tegemiseks

Mina, Karl Soosalu,

1. annan Tartu Ülikoolile tasuta loa (lihtlitsentsi) minu loodud teose „Kvaliteedi tagamise protsessid tarkvaraettevõttes Playtech Estonia näitel“, mille juhendajad on Heili Orav ja Jorma Pärn, reprodutseerimiseks eesmärgiga seda säilitada, sealhulgas lisada digitaalarhiivi DSpace kuni autoriõiguse kehtivuse lõppemiseni.

2. Annan Tartu Ülikoolile loa teha punktis 1 nimetatud teos üldsusele kättesaadavaks Tartu Ülikooli veebikeskkonna, sealhulgas digitaalarhiivi DSpace kaudu Creative Commonsi litsentsiga CC BY NC ND 3.0, mis lubab autorile viidates teost reprodutseerida, levitada ja üldsusele suunata ning keelab luua tuletatud teost ja kasutada teost ärieesmärgil, kuni autoriõiguse kehtivuse lõppemiseni.

3. Olen teadlik, et punktides 1 ja 2 nimetatud õigused jäävad alles ka autorile.

4. Kinnitan, et lihtlitsentsi andmisega ei riku ma teiste isikute intellektuaalomandi ega isikuandmete kaitse õigusaktidest tulenevaid õigusi.

Karl Soosalu

10.05.2022