Opinnäytetyö (AMK)

Tietotekniikka

Hyvinvointiteknologia

2017

Antti Ristolainen

TESTIAUTOMAATIO-PROSESSIN LUOMINEN

TURKU AMK
TURKU UNIVERSITY OF APPLIED SCIENCES

OPINNÄYTETYÖ (AMK) | TIIVISTELMÄ

TURUN AMMATTIKORKEAKOULU

Tietotekniikka | Hyvinvointiteknologia

2017 | 21

Antti Ristolainen

TESTIAUTOMAATIOPROSESSIN LUOMINEN

Tämän opinnäytetyön tarkoituksena oli ottaa käyttöön CGI:n Welfare yksikön Turun toimipisteeseen testiautomaatio. Tavoitteena testiautomaation käyttöönotolla oli vähentää työpanosta testaamiseen ja nopeuttaa testaamista. Tavoitteena oli automatisoida potilastietojärjestelmä Pegasoksen testausta, ja myöhemmin laajentaa automatisointia yksikön muihin tuotteisiin.

Projektin aluksi tutustuttiin nykyiseen testaukseen sekä ohjelmiston elinkaareen kehityksestä julkaisuun. Analysoimalla nykyinen testaus, saatiin kuva mitä testejä kannattaa automatisoida. Testiautomaatio-ohjelma, jolla automatisointi toteutetaan, tuli valita. Tätä varten tehtiin vaatimusmäärittely, jotta vertailu olisi helpompaa ja tasapuolista. Ohjelmia kartoitukseen otettiin 4, joista käyttöön valikoitui Ranorex.

Käytössä oleviin kehitysympäristöihin tutustuttiin, jotta saatiin kokonaiskuva mitä missäkin ympäristössä kehitetään, ja missä ympäristöissä testaamiselle on tarve. Kehitystyötä tehdään 8 ympäristössä, ja kaikissa näistä voi ohjelmaan tulla virheitä. Mitä nopeammin virheet löydetään, sitä helpompaa niiden korjaaminen on. Siksi siis testiautomaatiota tulee ajaa kaikkia ympäristöjä vasten. Ympäristöstä johtuvien virheiden vuoksi testiautomaatiolle haluttiin perustaa omat ympäristöt.

Testiautomaatiota voidaan kehittää ohjelmoiden, nauhoittamalla tehdyt toimenpiteet tai näiden kombinaatiolla. Ohjelmoimalla testejä niitä pystytään hallinnoimaan paremmin, ja niiden laajentaminen eri ympäristöihin on helpompaa. Nauhoituksella pystytään tekemään nopeammin, mutta ylläpito, ja laajennus ei ole niin helppoa. Siksi testit kannattaa ohjelmoida käsin.

Valmistuneet testit tulee testata huolellisesti useaan otteeseen. Testiautomaatio kulminoituu luotettavuuteen. Koska testejä ei käyttöönoton jälkeen useasti ainakaan ajeta käsin, tulee testeihin voida luottaa. Jos vääriä virhetilanteita tulee useasti, on testiautomaatio turhaa, kun virheet joudutaan manuaalisesti tarkastamaan.

Työn tuloksena valittiin ohjelma, jolla automatisointi tehdään. Testiautomaation analysoinnin myötä tiedettiin, mitkä testit kannattaa automatisoida ja miten ne toteutetaan. Ympäristöjen analysoinnin jälkeen tiedettiin, mitä ja missä ympäristöissä kannattaa testiautomaatiolla testata.

ASIASANAT:

Testiautomaatio,ohjelmistotestaus, projekti

BACHELOR´S THESIS | ABSTRACT

TURKU UNIVERSITY OF APPLIED SCIENCES

Information Technology | Health Informatics

2017 | 21

Antti Ristolainen

CREATING TEST AUTOMATION PROCESS

Click here to enter text.

This thesis subject is to start using test automation at The Welfare Department of the Turku branch of CGI. Objective of test automation was to reduce workload of testing and fasten testing process. Test automation project was started with automating patient information system Pegasos, and plan was to automate later other products of the department

In the beginning of project, current testing were familiarized with and cycle of software development. With analysing current testing, tests which are worth automating were found out. As the test automation process was developed from scratch, the software to be used also needed to be selected. For this purpose, a requirement list was created. Four different softwares were compared, and Ranorex was selected.

Current development enviroments were familiarized with, to have full picture of whats purpose of each enviroment, and in what enviroments testing is necessary. Pegasos is being developed in 8 enviroments and each enviroment is necessary for testing. Therefore, an ideal situation is to test all enviroments to find errors in software as soon as possible.

Test automation is developed by programming, recording completed actions on the software or a combination of these. By programming test cases, better control of testing can be achieved, and maintaining and distributing tests over different enviroments is easier. The recording method is quicker, but maintaining, updating tests and distributing over different enviroments is more complex. Due to the complexity of recording, the programming method is recommended.

The completed tests have to be tested multiple times. Test automation relies on reliability. After test are automated, they are rarely manually tested anymore, so automated tests have to be trusted. If too many false alerts are raised, trust in test automation decreases, and manual testing has to be carried out to verify that the software is functional. If manual testing has to be implemented to verify test automation errors, it will reduce the benefit of automation.

Based on the study, test automation software was selected. Test analysis was carried out, test cases worth automating were identified and developed. After analyzing development enviroments, enviroments where test automation should be executed were found.

KEYWORDS:

Test automation, software testing, project

SISÄLTÖ

KUVAT

1 JOHDANTO

Opinnäytetyön tavoitteena oli ottaa käyttöön CGI:n Welfaren Turun yksikössä testiautomaatio täydentämään nykyistä testausta sekä vähentämään työpanosta, jota testaukseen käytetään ja nopeuttamaan testausprosessia. Ensisijaisesti tavoite oli automatisoida potilastietojärjestelmä Pegasoksen testausta, mutta automaatiota halutaan laajentaa tulevaisuudessa kattamaan muitakin ohjelmia. Pelkästään uuden version julkaisussa käytettävä testausaika on noin 2 viikkoa, joten testiautomaatiolla voidaan saada huomattavat säästöt testaukseen käytettävässä ajassa. Tämän lisäksi testausta tehdään monessa eri muodossa, joten ajallisesti voidaan päästä suuriin säästöihin.

Opinnäytetyön toimeksiantajana toimii CGI Suomi Oy. CGI Suomi Oy kuuluu suurempaan kansainväliseen CGI-konserniin. CGI Suomi Oy aloitti toimintansa vuonna 2012, jolloin CGI osti Logican toiminnot ja liitti ne osaksi toimintaansa. Serge Godin ja André Imbeau perustivat CGI-emoyhtiön vuonna 1976 Kanadan Quebecissä. CGI:llä on maailmanlaajuisesti noin 68 000 työntekijää, joista Suomessa toimii noin 3000. (CGI 2017.)

Olin CGI:llä työharjoittelussa ohjelmistotestauksen tehtävissä, minkä jälkeen aloitin työskentelyn testiautomaatioprojektin parissa. Työharjoittelun parissa sain hyvän katsauksen ohjelmistotestaukseen ja opinnäytetyö testiautomaation parissa antoi hyvän näkökulman teknisemmän testauksen puoleen.

Opinnäytetyössä käydään läpi nykyiseen testaukseen mitä Welfare yksikössä käytetään. Testiautomaation osalta analysoidaan teoriaa sekä millaisia eri toteutusvaihtoehtoja testiautomaation toteuttamiselle löytyy. Koska testiautomaatio rakennetaan alusta asti tyhjästä, tuli myös tutustua eri ohjelmistovaihtoehtoihin. Ohjelman valitsemista varten tehtiin vaatimusmäärittely, jonka pohjalta eri ohjelmia oli helppo vertailla tasavertaisesti. Ohjelman valinnan jälkeen tutustuttiin testeihin ja siihen, miten testejä tullaan ajamaan, missä ympäristöissä ja kuinka usein. Ohjelmistokehitystä tehdään useassa ympäristössä, joten tuli miettiä, missä kaikkialla testiautomaatiolla halutaan testata. Testaustyö, mitä halutaan automatisoida, tuli myös analysoida ja valita.

2 TESTIAUTOMAATIO

Testiautomaatio tarkoittaa prosessia, jossa testiautomaatio-ohjelma suorittaa ennalta määritetyt suoritteet ohjelmistoa vasten. Testiautomaatiolla pystytään säästämään aikaa, kun version julkaisussa tehtävä regressiotestausta sekä muuta ohjelmistolle tehtävää testausta pystytään nopeuttamaan. Täten uusi ohjelmisto saadaan nopeammin julkaistua, ja arvokasta aikaa säästyy. Testauksen määrän lisääntyessä, mahdolliset ohjelmistovirheet jäävät aikaisemmin kiinni, jolloin niihin on helpompi päästä kiinni, kuin testauksen loppuvaiheessa jolloin ohjelmistoa ollaan julkaisemassa. Lisäksi myös nykymaailman ohjelmistokehityksessä kehitysympäristöjä ja ohjelmistoversioita on useita, on näiden kesken testien ajaminen myös helppoa. (TechTarget 2014)

2.1 Testiautomaation hyödyt

Testiautomaatiolla voidaan automatisoida nykyisiä käytössä olevia testejä, jolloin testaukseen käytettävä aika lyhenee. Vaihtoehtoisesti myös testauksen laajuutta voidaan kasvattaa. Koska testien ajaminen ei vaadi manuaalista työvoimaa, ja testejä voidaan ajaa kellon ympäri, saadaan laajempikin testikokonaisuus testattua nopeammin kuin pelkällä manuaalitestauksella. Julkaistava ohjelmisto saadaan julkaistua nopeammin ja laajemmin testattuna. (Utest 2016)

Manuaalitestauksessa syötettävän testidatan määrä on pieni ja sen vaihtelevuus ei ole suurta. Testiautomaatiolla voidaan kasvattaa testidatan määrää ja vaihtelevuutta, joka suoraan nostaa ohjelmiston laadunvarmistusta kattavammalla testauksella. Testidatan määrän kasvulla pystytään myös testaamaan, miten ohjelma käyttäytyy rasituksen alla, kun datan määrä on suuri. Manuaalitestauksessa on myös mahdollista, että testaaja tekee virheen. Tämä voi johtaa siihen, että testi menee virheellisesti läpi. Testaaja siis luulee testin onnistuneen, vaikka todellisuudessa testiajo ei ole vastannut testitapausta. Toinen vaihtoehto on, että testaaja testaa ohjelmaa väärin ja päätyy erheellisesti virhetilanteeseen. Testiautomaatiossa testit pystytään aina ajamaan yhdenmukaisesti, jolloin testien luotettavuus nousee. (Utest 2016; Guru99 2017)

Testausta pystytään myös laajentamaan helposti kattamaan eri laite ja käyttöjärjestelmä kombinaatioita. Mobiilisovellusten testauksessa päätelaitteita löytyy eri käyttöjärjestelmillä, pääasiassa Androidilla, Ios:lla tai Windows Phonella. Käyttöjärjestelmistä on monia

versioita jonka lisäksi myös asennetut ohjelmat laitteille sekä laitteiston konfiguraatiot vaihtelevat. Sama tilanne toistuu myös tietokoneilla; eri käyttöjärjestelmiä, komponentteja ja asennettujen ohjelmien kombinaatioita on lukemattomasti, joten manuaalitestauksella jokaista kombinaatiota ei pystytä testaamaan. Ohjelmistojen kehitysympäristöjä saattaa olla myös monia, ja julkaistujen ohjelmistojen versioita saatetaan pitää vielä korjauksia varten ylhäällä. Myös näiden ympäristöjen testaus onnistuu tehokkaasti. (Utest 2016; Guru99 2017)

2.2 Testiautomaation toteutus

Testiautomaatio toteutetaan yleensä joko nauhoittamalla, ohjelmoimalla tai näiden yhteistyöllä. Testien nauhoittamisesta puhuttaessa testaaja suorittaa testitapauksen mukaisesti testin testiautomaatio ohjelman nauhoittaessa, ja testin lopuksi tehdyt toimenpiteet ovat tallennettuna ja toistettavissa. Tällä tavalla testitapauksia pystytään automatisoimaan nopeasti, ja menetelmä sopii hyvin yksinkertaisten testien automatisointiin. Kuitenkin testitapausten monipuolistuessa, tai testitapauksen kasvaessa nauhoitus asettaa omat haasteensa. Nauhoitettujen testien hallinnointi ja ylläpito tuovat myös lisähaastetta. Nauhoittaessa ohjelma ei osaa käyttää optimaalisia tunnistetietoja elementeille, jolloin testit eivät välttämättä toimi muissa ympäristöissä, tai vaativat manuaalista työvoimaa näiden sovittamiseksi muihin ympäristöihin.

Ohjelmointi menetelmää käytettäessä testit automatisoidaan ohjelmallisesti. Yleiset ohjelmointikielet, kuten Java ja C# ovat yleisesti tuettuina, ja näitä täydennetään testiautomaatio-ohjelmistojen omilla kirjastoilla. Tällä tarkoitetaan, että ohjelman puolelta on valmiina esimerkiksi hiiren painallus objekteille. Ohjelmoimalla testit pystytään ohjelmoimaan halutuiksi, ja valinnan varaa suunnitteluun on paljon enemmän. Monimutkaisempien testien automatisointi on helpompi kuin nauhoittaessa, ja ohjelmallisesti pystytään hallitsemaan koko testiä.

3 NYKYINEN TESTAUS

Ohjelmistotestaus on osa ohjelmistokehitystä. Helposti voidaan olettaa, että ohjelmistokehitys on vain pelkästään ohjelmiston koodausta. Tämä olettamus on väärä, ja ohjelmistotestaus on yhtä tärkeä osa ohjelmistokehitystä kuin itse koodaus. ”IEEE:n (1990) määritelmän mukaan ohjelmistotestaus on aktiviteetti, jossa määritellyillä ehdoilla suoritetun ohjelman tai sen osan tuloksia tarkastellaan tai tallennetaan, ja sen toimintaa arvioidaan.” (Jyväskylän yliopisto, Systeemiset oppimisratkaisut 2017).

Ohjelmistotestauksen tavoitteena on varmistaa, että ohjelmistosta ei löydy ohjelmistovirheitä. Testauksella myös pystytään osoittamaan ohjelmistosta löydetyt virheet, joita ilman testausta ei löydettäisi. Testauksella myös varmistetaan, että ohjelmisto tekee sen mitä siltä halutaan, täyttää ohjelmiston vaatimukset, ja varmistaa ettei ohjelmisto tee sellaisia toimenpiteitä joita siltä ei odoteta. Testaamalla pystytään tarjoamaan laadukas ohjelmisto. Testaaminen aiheuttaa kustannuksia muun muassa henkilöstökuluina. Jos ohjelmistoa ei kuitenkaan testattaisi, aiheuttaa se suuremmat kustannukset asiakastyytymättömyydellä virheellisiin ohjelmiin. Ohjelmistovirheet ovat myös kustannukseltaan halvempi korjata kehitysvaiheessa, kuin siinä vaiheessa, jossa asiakas käyttää jo ohjelmistoa. (ISTQB exam certification 2016; Testing Excellence 2010)

Jotta testiautomaation käyttöönotto olisi mahdollisimman helppoa ja tukisi mahdollisimman hyvin nykyistä testausta, tutustuttiin aluksi nykyiseen käytössä olevaan testaukseen.

3.1 Regressiotestaus

Regressiotestaus tarkoittaa testausta, jolla varmistutaan, että ohjelmisto toimii tehtyjen muutoksien jälkeen. Vaikka komponentti tasolla ohjelma toimii, se ei tarkoita, että ohjelmisto toimii isommassa mittakaavassa, johon regressiotestaus iskee. Regressiotestausta suoritetaan ohjelmaan tehtyjen muutoksien jälkeen, jotka voivat olla esimerkiksi ohjelmistoon tehtyä uutta toiminallisuutta, tai vaihtoehtoisesti myöskin olla aikaisempien ohjelmistovirheiden korjausta tai vaikka järjestelmäparametrien muokkaamista.

Regressiotestauksessa ajetaan ennalta kirjoitettuja testejä, jotka sisältävät testauksessa tehtävät syötteet sekä jokaisen syötteen jälkeisen odotetun tuloksen. Regressiotestausta

tehdään koko ohjelmistolle, mutta sitä voidaan myös supistaa, jos pystytään rajaamaan tehdyt muutokset esimerkiksi komponenttien mukaan. Regressiotestaus sopii toistuvuuden vuoksi hyvin automatisoitavaksi. (Kuopion yliopisto 2004; Wikipedia 2017)

3.2 Yksikkötestaus

Yksikkötestauksessa kehitystiimin testaa, että tehty toiminallisuus täyttää vaatimukset tai virhekorjaus korjaa havaitun virheen. Testaus tehdään vaatimuksia vasten, ja testit ovat luonteeltaan lyhyitä ja yksinkertaisia. Testeistä on vaikea saada kokonaisuuksia automatisoitavaksi, mutta yksittäisten komponenttien tarkempaan automatisointiin testit soveltuvat.

3.3 End to end -testaus

End to end -testauksessa testataan liitettävän komponentin tai osuuden vaikutusta koko ohjelmistoon. Testauksen ideana on kulkea loppukäyttäjän näkökulmasta koko ohjelmapolku uuden ohjelmiston osalta. Tällöin havaitaan, toimiiko liitetty ominaisuus yhdessä ohjelmiston muiden osien kanssa. Uusi toiminallisuus voi esimerkiksi tuottaa dataa, jota ohjelmiston jo käytössä oleva osio käyttää. Hyvin kirjoitetut end to end -testit vastaavat pitkälti regressiotestejä, joten end to end testejä voidaan myös ottaa myöhempään käyttöön regressiotestauksessa ja siten myös automatisoitaviksi testeiksi. (Software testing help 2017)

3.4 Hyväksyntätestaus

Hyväksyntätestauksessa keskitytään loppukäyttäjän vaatimusten täyttämiseen. Testaus toteutetaan black box -testauksena, jolloin käyttäjä ei keskity eikä välttämättä tiedä koodin toiminnasta, vaan keskittyy ohjelmiston ulkoisiin ominaisuuksiin. Hyväksymistestauksessa käytetään yleensä asiakasta, tai vaihtoehtoisesti muuta loppukäyttäjään verrattavaa henkilöä. Koska testaus keskittyy käyttäjän näkökulmasta ohjelman käytettävyyteen, ei testeistä saa automatisoitavia testitapauksia. (Software testing class 2012)

4 TESTIAUTOMAATIO OHJELMAN VAATIMUKSET

Ennen ohjelmien vertailun aloittamista tehtiin vaatimusmäärittely. Vaatimusmäärittelyyn mietin, mitä ominaisuuksia ohjelmalla tulee olla, jotta sen käyttö on mielekästä. Vaatimusmäärittelyn avulla pystytään tehokkaammin vertailemaan eri ohjelmia, ja vertailua tehdessä pystytään helpommin arvioimaan ohjelmistojen väliset erot vaatimuksiin verraten.

4.1 Uniface-tuki

Ohjelmasta tulee löytyä Uniface -ohjelmointikielen tuki. Pegasoksessa monet osiot on ohjelmoitu Unifacella. Jollei Uniface tukea ole, suuri osa Pegasoksen toiminnallisuudesta jää automaatiokehityksen ulkopuolelle. Tämä ei ole järkevä valinta, koska kun projekti aloitetaan puhtaalta pöydältä, on mahdollisuus valita automaatio-ohjelma joka kattaa tämän vaatimuksen.

4.2 Java-tuki

Ohjelmasta tulee löytyä Java -ohjelmointikielen tuki. Java on yleinen maailmalla tuettu ohjelmointikieli. Tämän vaatimuksen täyttäminen ei ole vaikeaa, sillä ohjelmointikielen yleisyyden takia se on oletusarvoisesti tuettuna. Pegasoksessa on myös käytetty Java ohjelmointikieltä. Jos Java ohjelmointikielen tukea ei löydy, Pegasoksesta suuri osa, ellei jopa kokonaan, jää automaatiokehityksen ulkopuolelle. Myöskään automaation laajentaminen ei onnistu mahdollisesti muihin projekteihin Pegasoksen jälkeen, koska erittäin todennäköisesti Javaa on käytettynä.

4.3 Windows Server 2008 R2 -tuki

Windows Server 2008 R2 -tuki mahdollistaa automaatio-ohjelman asentamisen virtuaalikoneelle. Virtuaalikoneella tarkoitetaan virtuaalisesti ajettua tietokonetta, johon voidaan fyysiseltä koneelta ottaa yhteys. Pegasoksen kehitysympäristöt, joissa myös automaa-

tiotestit tulee ajaa, toimivat Windows Server 2008 R2-käyttöjärjestelmällä. Näin testiympäristöistä saadaan yhdenmukaiset, ja ympäristöistä johtuvat virhetilanteet ja mahdollisen komplikaation riskejä pystytään vähentämään.

4.4 Jenkins-tuki

Pegasoksen kehitysympäristöt kääritään kerran vuorokaudessa ajastetusti. Jotta ohjelmistovirheisiin päästään mahdollisimman nopeasti kiinni, on tärkeää pystyä ajamaan testit heti käärinnän jälkeen. Pegasoksen käärinnät tehdään Jenkinsillä, joten myöskin testiautomaatioiden käynnistys voidaan luontevasti tehdä Jenkinsillä. Jollei Jenkins tukea löydy, testien ajaminen tuottaa enemmän vaivaa, sekä ohjelmistovirheet, että ohjelmiston virhetilanteet löydetään vasta myöhemmin.

4.5 Testiajojen raportointi

Testiajojen raportointi täytyy olla tuettuna. Testiajojen raportoinnilla testien ajoista pysytään kartalla, ja tiedetään, mitkä testit on ajettu ja milloin. Lisäksi nykyinen manuaalitestaus on raportoitu ja dokumentoitu, jotta lääkintälaitelainsäädännön vaatimus dokumentoinnista säilyy. Siksi siis testiautomaationkin tulee tukea raportointia. Jollei testiajojen raportteja ole tuettu, on hankalaa tai mahdotonta osoittaa mitä testejä on milloinkin testattu.

4.6 Raporttien muokattavuus

Raportit voidaan muokata mieleisiksi, jolloin niihin saadaan sellainen tieto mikä on tärkeää. Esimerkiksi testiajoista voidaan koota statistiikkaa, kuinka monta kertaa kyseinen testi on ajettu ja montako kertaa kyseinen testi on päätynyt virheeseen. Tällöin pystytään havainnoimaan, onko esimerkiksi kyseinen testi virhealttiimpi. Vaihtoehtoisesti voidaan myös huomata, että testi on ensimmäisen kerran ajautunut virheeseen, jolloin on todennäköistä, että ohjelmistossa on virhe

4.7 Testien modulointi

Jotta uusien testien suunnittelu olisi helpompaa, pitää testit pystyä pilkkomaan pienemmiksi moduuleiksi. Moduuleista pystytään rakentamaan uusia testejä, jolloin kaikkea tekemistä ei tarvitse aloittaa puhtaalta pöydältä, vaan aiempaa toiminallisuutta pystytään hyötykäyttämään. Jollei testejä pysty pilkkomaan moduuleiksi, turhaa työtä joudutaan tekemään samojen asioiden toistamiseen testistä toiseen.

4.8 Data Driven -testaus

Data driven -testauksella pystytään syöttämään isompi skaala syötteitä testille ja voidaan testata, käyttäytyykö ohjelma odotetusti. Testaukseen voidaan määrittää esimerkiksi excel -tiedosto, johon syötetään esimerkiksi käyttäjätunnuksia sekä salasanoja, joita käytetään kirjautumiseen. Testi ohjelmoidaan käyttämään excelin käyttäjätunnuksia ja salasanoja, ja voidaan havaita, miten esimerkiksi ohjelma käyttäytyy erikoismerkeillä.

5 TESTIAUTOMAATIO OHJELMAN VALINTA

Jotta selvitystyö pystyttäisiin pitämään tarpeeksi rajattuna, niin selvitykseen valittiin 4 ohjelmaa. Ohjelmat valittiin projektin aloituspalaverissa. Seuraavat ohjelmat valittiin selvitykseen mukaan:

- Ranorex
- IBM Rational Functional Tester
- HP Unified testing
- SilkTest.

IBM Rational Functional tester on ollut yrityksen toisen tuotteen testauksessa mukana. Ohjelma otettiin mukaan selvitykseen, jotta tiedetään, onko ohjelmalla potentiaalia tulla laajempaan käyttöön. HP Unified testing valikoitu vertailuun, sillä se on valittu uudeksi työvälineeksi testauskonsultointiyksikössä. Silktest valikoitui palaverissa käydyn keskustelun myötä kuten Ranorexkin.

Selvitystyötä varten Pegasoksesta luotiin jokaiselle ohjelmalle oma testiympäristö, johon kuhunkin asennettiin normaalin Pegasos-työaseman ohjelmistot sekä testattava ohjelmisto. Jokaiselle ohjelmalle luotiin oma ympäristö siksi, että mahdolliset komplikaatiot eri ohjelmien välillä vältettäisiin, ja testaus olisi mahdollisimman autenttinen. Testattaville ohjelmille kullekin annettiin 4 päivää aikaa tutustumiseen. Tavoitteena oli tutustua kuhunkin ohjelmaan ja tutkia miten vaatimukset täyttyvät ohjelman osalta.

Vaatimuksia oli runsaasti, joten vaatimuksia täytyi myös priorisoida. Tärkeimmäksi vaatimukseksi nostettiin tuettavat tekniikat. Ilman laajaa tukea ohjelmalla ei pysty tekemään testiautomaatiota. Lisäksi myös käyttöjärjestelmätuki oli korkealla prioriteetilla, jotta ympäristöt olisivat yhtenäisiä. Testien raportointi nousi kolmanneksi tärkeäksi, jotta testit pystytään raportoimaan ja todentamaan, mitä on testattu.

Vaatimusten ohella huomiota kiinnitettiin ohjelman käytettävyyteen. Ohjelman tuli olla helposti käytettävä. Hankalan ohjelman käyttäminen ja opettelu pitkällä aikavälillä ei ole hyvä idea, kun on mahdollisuus tyhjältä pöydältä aloittaa projekti ja valita ohjelma vapaasti. Käytettävyyden lisäksi huomiossa oli myös ohjelmoitujen testien sekä nauhoitettujen testien liitettävyys. Kaikki testit eivät välttämättä tarvitse ohjelmointia, joten jos nauhoitusta sekä ohjelmointia voidaan kätevästi liittää toisiinsa, testejä voitaisiin automatisoida nopeammin. Testiajojen suunnittelun helppous oli myös tarkastelussa. Jos testit

kootaan moduuli ajatuksella, niin miten moduuleista pystytään kokoamaan ajettava testiajo? Jos testiajon suunnittelu ei ole helppoa, tuo se ylimääräistä työtä, jos halutaan ajaa spesifisti tietyt testit.

5.1 Testiautomaatio-ohjelman valinta

Valinta päätyi Ranorexiin. Ranorex esiteltiin loppupalaverissa ja sai hyväksynnän käyttöönotolle. Ranorexia esitettiin valittavaksi, koska ohjelma oli selvityksessä olleista helpoin opittava ja täytti asetetut vaatimukset, joita ohjelmalle asetin.

Ranorexissa luodaan uusi projekti, jota Ranorexissa kutsutaan solutioniksi. Solutionilla on yksi testiajo pohja, johon kootaan ajettavat testit. Testiajolle kootaan kansioita, jotka voidaan nimetä esimerkiksi komponentin mukaan. Kansion alle luodaan testit, joita Ranorexissa kutsutaan test caseiksi. Test casen alle raahataan ohjelmoidut tuotokset ja nauhoitetut tuotokset. Ohjelmointeja Ranorexissa kutsutaan code moduleiksi, ja nauhoituksia recording moduleiksi. Test casen sisällä pystytään rajattomasti liittämään moduuleita, kunhan moduulit on suunniteltu siten, että ne jatkavat siitä, mihin aiempi moduuli on jäänyt. Testiajoa suunniteltaessa, valitaan mitkä kansiot, tai testit halutaan ajaa. Testiajon jälkeen raportti testiajosta tulostuu. Raportilta on nähtävillä ajetut testit, sekä ovatko testit ajettu onnistuneesti läpi. Virheeseen päätyneet testit näkyvät punaisella selkeästi eroteltuina.

Ranorex tuki lisäksi myös laajasti eri tekniikoita. Tämä mahdollistaa Ranorexin laajan käytön taustaltaan eri sovelluksissa, joten ohjelmaa voidaan käyttää myös laajasti muiden automatisointi projektien parissa. Testiautomaation laajentaminen yrityksen eri tuotteisiin on siis mahdollista. Lisäksi Ranorexilla on natiivin mobiilitestauksen tuki Androidille ja Ios:lle, joka oli hyvä erottava tekijä.

6 KEHITYSYMPÄRISTÖT

Pegasosta kehitetään useassa kehitysympäristössä. Onkin tärkeää hahmottaa, mitä kehitystä tehdään missäkin ympäristössä. Tällöin kokonaisuuden hahmottaminen on helpompaa, ja voidaan todeta missä ympäristöissä on tarve testiautomaatiolle.

6.1 Testiympäristöt

Pegasos kehitystä tehdään neljässä eri kehitysympäristössä:

- Trunkissa
- Nextissä
- julkaistavassa versiossa
- edellisessä julkaistussa versiossa.

Jokaisesta kehitysmpäristöstä on vielä Build-, Dailytesti- ja Systeemitesti-versiot. Build ympäristössä ei tehdä yleistä testausta. Kuvan 1 mukaisesti, ympäristöjä joissa testausta tehdään, on 8 kappaletta.

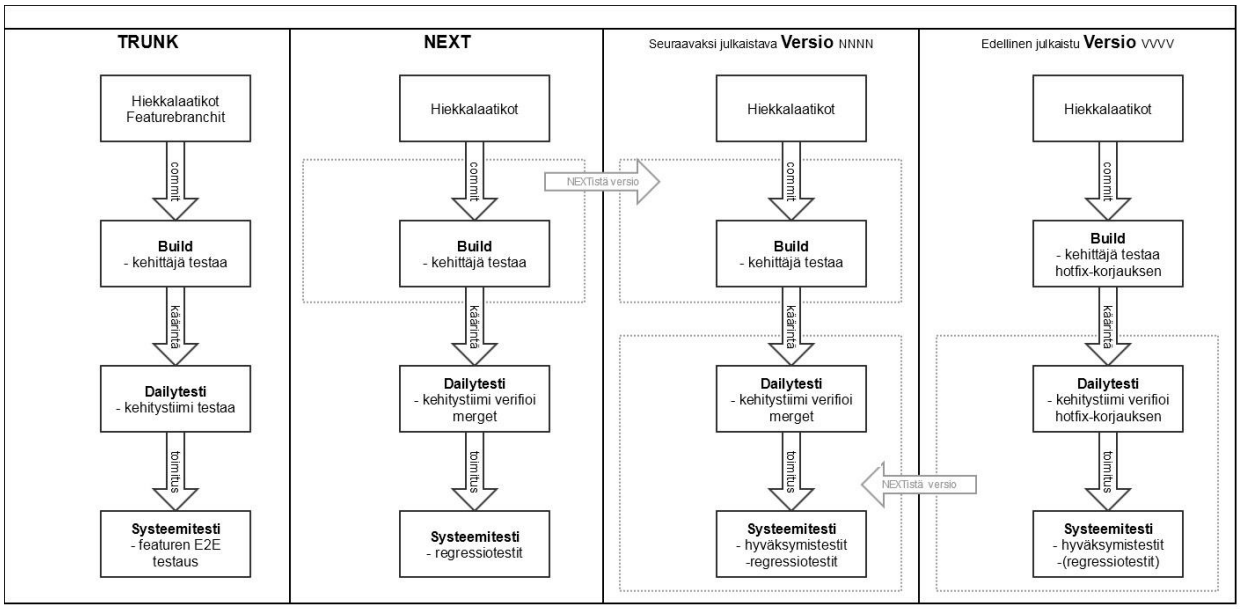
Trunk-kehitysympäristöön tuodaan ensiksi ohjelmistoon tuotavat uudet toiminallisuudet. Kehittäjä ensiksi testaa Build-ympäristössä uuden toiminallisuuden. Jos uusi toiminallisuus todetaan kehittäjän toimesta toimivaksi, se siirtyy joka öisen käärinnän mukana dailytesti ympäristöön. Dailytestissä kehitystiimi testaa, että toiminallisuus on toimiva. Tästä seuraava kohta on Systeemitestin testaus. Systeemitestiympäristö kääritään tarvittaessa, sekä sprinttien jälkeen. Systeemitesti ympäristössä toiminallisuudelle suoritetaan end to end -testaus. Jos end to end -testaus menee läpi, tiimi mergeää eli liittää tehdyn toiminallisuuden next ympäristöön.

Nextin buildissa kehittäjä varmistaa mergen onnistumisen. Nextin dailytesti kääritään joka yö kuten trunkin dailytesti. Dailytestissä kehitystiimi varmistaa ja verifoi mergen toimivuuden. Nextiin tuodaan myös ohjelmistovirheiden korjaukset, jotka testataan samalla logiikalla. Nextin systeemitestissä tehdään regressiotestaus.

Tästä siirrytään julkaistavaan versioon. Kun Nextissä oleva versio todetaan valmiiksi julkaisuun, nextin tilanne jäädytetään ja sen pohjalta lohkaistaan uusi versio. Version buildissa kehittäjät testaavat mergejensä sekä ohjelmistovirheiden korjausten

toimivuuden. Tästä siirrytään dailytesti ympäristöön, jossa kehitystiimi verifioi merget. Tämän jälkeen systeemitesti ympäristössä suoritetaan hyväksymistestaus sekä version regressiotestaus. Hyväksymistestauksen ja regressiotestauksen jälkeen versio on valmis julkaistavaksi.

Julkaistun version lisäksi vanhoista versioista on ympäristöt käynnissä. Build ympäristössä kehittäjä testaa hotfixin toimivuuden. Tästä siirrytään dailytestiin jossa kehitystiimi verifioi hotfixin. Systeemitesti ympäristössä hotfixeille tehdään hyväksymistestaus sekä tarvittaessa regressiotestaus.



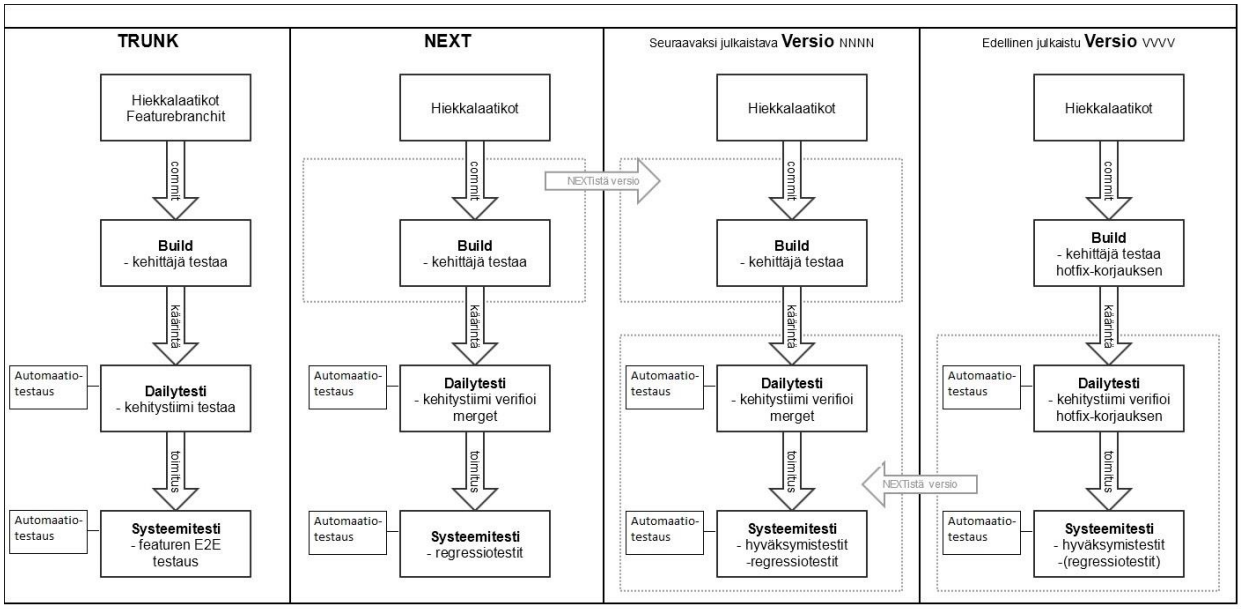
Kuva 1. Kehitysympäristöt

6.2 Testiautomaatio-ympäristöt

Jotta testien ajaminen on helppoa, tulee miettiä miten ja missä ympäristöissä testejä ajetaan. Jos testejä ajetaan samoissa ympäristöissä mitä käytetään manuaalitestaukseen, voi syntyä komplikaatio tilanteita. Myös tietokantojen muutokset voivat vaikuttaa testien suorittamiseen. Jos testi on ohjelmoitu toiminaan tietyllä ohjelmiston asetuksella, ja tätä asetusta muutetaan, testi ei enää toimi. Testejä on mahdoton suunnitella siten, että ne kattaisivat kaikki eri asetus kombinaatiot, koska eri kombinaatioita on runsaasti ja täten työmäärä olisi suuri. Testit tulee siis ajaa omassa ympäristössä. Tällöin vältytään turhalta testien ylläpidolta, sekä voidaan sulkea pois asetuksien muutoksista johtuvat virheelliset hälytykset.

Testiautomaatio ympäristöt tulee päivittää samaa tahtia kuin normaalit ympäristöt. Tällöin uusimmat ohjelmistoon tuodut muutokset ovat heti testattavissa. Testiautomaatio-ympäristöt tulee ajastaa päivitettäväksi Jenkinsin avulla kuten normaalit ympäristöt. Normaalit kehitysympäristöt päivittyvät kello 1.00 aamuyöstä. Ympäristöjen käärinnän ja päivityksen kesto vaihtelevat riippuen ohjelmistoon tuotujen muutoksien määrästä, joka tulee ottaa huomioon testiautomaatio ympäristöjen päivittämisessä. Ympäristöjen päivityksen jälkeen testit ajetaan.

Jotta mahdollisiin defekteihin päästään mahdollisimman nopeasti kiinni, tulee testiautomaatiota ajaa Trunkin, Nextin sekä version Daily- ja Systeemitesteissä kuvan 2 mukaisesti. Eri ympäristöissä kehitetään eri tasolla ohjelmistoa, ja yhtä lailla Trunkissa kehitetty uusi toiminnallisuus voi hajottaa Pegasoksen toiminallisuutta yhtä lailla kuin Nextissä tehty ohjelmistovirheen korjaus. Testiautomaation potentiaali saadaan myöskin parhaiten esille, kun testejä pystytään ajamaan mahdollisimman paljon eri ympäristöissä.



Kuva 2. Testiautomaatio ympäristöt

7 AUTOMATISOITAVAT TESTIT JA NIIDEN TOTEUTUS

Testiautomaatio ohjelman ja ympäristöjen jälkeen tulee miettiä, mitä testejä kannattaa automatisoida. Jotta testiautomaation käyttöönotosta saadaan paras mahdollinen hyöty, tulee valita sellaiset, testit joista saadaan paras hyöty. Jos testi ajetaan harvoin, sitä ei ole viisasta automatisoida. Myös testit, joissa ei ole laajaa testidataa käytössä, eivät ole optimaalisimpia automatisoitavaksi. SmartBearin mukaan paras hyöty saadaan:

- Testeistä joita ajetaan useamman ohjelmistoversion kesken
- Testit, jotka ovat alttiimpia ihmisen tekemille virheille
- Testit, jotka vaativat laajaa testidataa
- Toiminallisuus, jota käytetään usein ja sisältää suuren riskin toimimattomuudella
- Testit, joita ei pystytä manuaalisesti ajamaan
- Testit, joita ajetaan eri laitteistoilla, käyttöjärjestelmillä tai asetuksilla
- Testit, jotka vievät paljon työvoimaa manuaalisesti testattuna.

Aiemmin nykyisestä testauksesta tehdyn selvityksen perusteella, regressiotestit vastaavat parhaiten edellä tehtyjä kuvauksia. Regressiotestejä ajetaan usein ja useassa eri ympäristössä. Regressiotestejä on käytössä runsaasti, joten joukkoon mahtuu hyviä automatisoitavia testejä jonka lisäksi automaatioon huonosti sopivia. Testit tulee analysoida, jotta optimaalisimmat testit automatisoidaan. Automatisointi kannattaa aloittaa sellaisista regressiotesteistä, joissa käytetään tai voidaan käyttää laajasti eri testidataa, ohjelmiston osa-alueissa joiden toimimattomuus aiheuttaa vakavan vaaratilanteen sekä testeistä jotka vaativat paljon työtä manuaalitestauksessa. (Smartbear, 2017.)

7.1 Testien toteutus

Testien valinnan jälkeen, luodut testit tulee jakaa tarpeeksi pieniin osiin. Jos kaikki testit kootaan yhdeksi isoksi testiksi, on testien hallinnointi vaikeaa, ja virhetilanteissa virheen paikantaminen koodista on myös vaikeaa. Testit tulee myös suunnitella siten, että käyttöliittymän muutokset ovat mahdollisimman vähäiset. Testiautomaatiossa käytetään tunnistetietoja, joilla tunnistetaan ohjelmiston objektit. Jos tiedot muuttuvat, riippuen testin suunnittelusta, testi voi päätyä virheeseen. Ranorexilla objektit on mahdollista tallentaa erilliseen kirjastoon, jonka käyttö on suositeltavaa. Tällöin kootut objektit ovat helposti

selattavissa kootusti. Kirjastoja voidaan luoda useita, jolloin ohjelmiston eri osuuksille voidaan luoda omat kirjastot.

Sitä mukaan, kun testejä valmistuu, on ne hyvä katselmoida, jotta testit vastaavat testitapausta. Katselmoinneista on hyvä pitää pöytäkirjaa, jotta voidaan todentaa, että testi on katselmoitu ja hyväksytty käyttöön. Valmistuneet testit tulee myös testata huolella, sillä testiautomaatio kulminoituu luotettavuuteen. Testien tulee olla luotettavia. Vaikka testi on ohjelmoitu painamaan tietyt painikkeet ja tekemään tietyt asiat, voi testin ajossa silti tulla ongelma, kun testiajoja on runsaasti takana. Luotettavuuden varmistamiseksi testit tulee ajaa moneen kertaan, jolloin voidaan havaita nuo ongelmakohdat, ja joko muokata testiä tai lisätä erilaisia lisävarmistuksia kyseisiin kohtiin, jotka tuottavat virhetilanteita.

8 YHTEENVETO

Tavoitteena oli vähentää manuaalisen testauksen viemää työvoimaa ja nopeuttaa testausta. Testiautomaatiolla pystytään automatisoimaan testausta, ja se pääsee oikeuksiinsa toistuvassa testauksessa sekä suurella testidatalla tehtävässä testauksessa. Testiautomaatio haluttiin siis ottaa käyttöön. Tätä varten kartoitettiin testiautomaatio-ohjelmia, joista käyttöön valikoitui Ranorex. Nykyiset testauksessa käytetyt ympäristöt haluttiin pitää erillään, jotta ohjelmista ei synny komplikaatioita. Lisäksi myös ympäristöjen asetuksia muutetaan usein testauksien yhteydessä, mikä tuo ongelmia. Testiautomaatiolle perustettiin tämän vuoksi vastaavat ympäristöt.

Kun regressiotestejä valitaan automatisoitavaksi, tulee ottaa huomioon testien vaatima työmäärä manuaalitestauksessa verrattuna työmäärään, joka automatisointiin kuluu. Lyhyitä ja helppoja testejä ei ole järkevää ainakaan aluksi automatisoida. Testit, jotka vaativat suurta työvoimaa ja käyttävät tai voivat käyttää laajaa testidataa, kannattaa priorisoida korkealle.

Kun testien automatisointityö aloitetaan, tulee kiinnittää huomioita yhtenäiseen ulkoasuun, jotta ylläpito ja muokkaus ovat helpompia. Valmistuneet testit tulee testiajaa useaan otteeseen, jotta niiden toimivuudesta voidaan olla varmoja.

Testien luotettavuus tulee myös varmistaa. Jos testit eivät ole luotettavia, eli päätyvät usein virheeseen, joka ei johdu ohjelmistosta, menettää automaatio hyötynsä. Tällöin joudutaan useasti tarkastamaan manuaalisesti, onko ohjelmisto oikeasti hajonnut, mikä tietää manuaalista työvoiman käyttöä. Jotta luotettavuudesta voidaan varmistua, olisi testit hyvä ajaa useasti peräkkäin. Tällöin saadaan tilastotietoa testin luotettavuudesta, ja virheisiin voidaan puuttua.

LÄHTEET

CGI 2017. Viitattu 12.4.2017 https://www.cgi.com.

Guru99 2017. Automation testing tutorial: Process, Planning & Tools. Viitattu 10.5.2017 http://www.guru99.com/automation-testing.html.

TechTarget Software Quality 2014. Automated software testing. Viitattu 15.4.2017 http://search-softwarequality.techtarget.com/definition/automated-software-testing.

Jyväskylän yliopisto 2017. Testaus lyhyesti. Viitattu 17.4.207 http://smarteducation.jyu.fi/projektit/systech/Periaatteet/suunnittelun-periaatteet/testaus/testaus-lyhyesti.

Kuopion yliopisto 2004. Regressiotestauksen tehostaminen. Viitattu 3.4.2017 http://cs.uef.fi/uku/tutkimus/Teho/RegressioselvitysJuha04.pdf.

Utest 2016. Top 15 Benefits of Automated Testing Tools. Viitattu 8.5.2017 https://www.utest.com/articles/top-15-benefits-of-automated-testing-tools.

Software testing class 2012. User Acceptance Testing: What? Why? & How?. Viitattu 12.5.2017 http://www.softwaretestingclass.com/user-acceptance-testing-what-why-how/.

Software testing help 2017. Why End to End Testing is Necessary and How to Perform It? Viitattu 12.5.2017 http://www.softwaretestinghelp.com/what-is-end-to-end-testing/.

Testing Excellence. Why do we Test ? What is the Purpose of Software Testing ?. Viitattu 12.5 http://www.testingexcellence.com/why-do-we-test-what-is-the-purpose-of-software-testing.

Wikipedia 2017. Regressiotestaus. Viitattu 3.4.2017 https://en.wikipedia.org/wiki/Regression_testing.

ISTQB exam certification 2016. Why is software testing necessary? Viitattu 14.5.2017 http://istqbexamcertification.com/why-is-testing-necessary/.

Smatbear 2017. Six tips to get started with automated testing. Viitattu 13.5.2017. Saatavilla sähköisesti osoitteessa http://www2.smartbear.com/rs/smartbear/images/6Tips.pdf.