Tampereen ammattikorkeakoulu

Jatkuva testaus sovelluskehityksessä

Automaatiotestauksen käyttöönotto Accountor HR Solutions Oy:ssä

Pirita Riutta

OPINNÄYTETYÖ
Huhtikuu 2019

Tietojärjestelmäosaaminen, ylempi AMK

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tietojärjestelmäosaaminen, ylempi AMK

RIUTTA, PIRITA:
Jatkuva testaus sovelluskehityksessä
Automaatiotestauksen käyttöönotto Accountor HR Solutions Oy:ssa

Opinnäytetyö 76 sivua, joista liitteitä 5 sivua
Huhtikuu 2019

Opinnäytetyö toteutettiin Accountor HR Solutions Oy:lle, joka on henkilöstö- ja palkkahallinnon ratkaisuihin keskittynyt asiantuntija- ja teknologiatalo. Yritys on kasvanut nopeasti, ja muutamassa vuodessa tuoteportfolio on laajentunut. Versiosyklien kiihtyessä ja tuoteportfolion laajetessa sekä haarautuessa manuaalinen testaus ei yksistään kuitenkaan enää riitä tuotteen laadun varmistamiseen aikataulussa. Tästä lähti muutossysäys automaatiotestauksen käyttöönottoon tuotekehityksessä.

Opinnäytetyön tavoite oli kehittää ja parantaa Mepco-tuoteperheen laatua ja kehitysprosessia sekä varmistaa toimintavarmuutta eri versioiden välillä. Tarkoitus oli ottaa käyttöön automaatiotestaus, luoda ohjeistukset ja toimintamallit testauksen jalkautukseen sekä testata luodut ohjeistukset ja suositukset käytännössä. Tutkimusmenetelmänä oli toimintatutkimus. Tietopohjana on kartoitukset yrityksen tilanteesta, tarpeista ja kehitysprosessista, aiheeseen liittyvä kirjallisuus ja kirjoittajan henkilökohtainen kokemus yrityksen toimintatavoista, tarpeista ja testauksen automatisoinnista.

Opinnäytetyön tuloksena saavutettiin automaatiotestausvalmius kohdeorganisaatiossa. Käytettäviin työkaluihin tehtiin soveltuvuusselvitys (Proof of Concept) ja laadittiin ohjeistukset. Osana testidatan hallintaa EU:n yleisen tietosuoja-asetuksen (EU 2016/679) soveltaminen testidatan suhteen tarkastettiin ja luotiin ohjeistus, jolla voidaan paitsi arvioida suojaustoimien tarvetta, myös varmistaa suojaustoimien kattavuus. Käyttöönotossa havaitut haasteet ja hyväksi todetut käytännöt käydään läpi muutoksenhallinnan näkökulmasta.

Jatkotoimenpiteenä ehdotetaan automaatiotestauksen jalkauttamista laajempaan käyttöön. Jaettu vastuu tuotekehityksen ja testauksen laadusta tarkoittaa, että osana jalkautusta testauksen vaatimia taitoja ja pätevyyksiä koulutetaan kaikille tuotekehityksessä, ei vain nimenomaisesti testaajille. Resursoinnin kannalta suositellaan automaatiotestauksen roolin asettamista prosessiin, jotta automatisoinnille varataan riittävät resurssit.

Asiasanat: automaatiotestaus, testidatan hallinta, datan anonymisointi, pseudonymisointi

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Master's Degree Programme in Information Systems Competence

RIUTTA PIRITA:
Continuos Testing in Software Development
Introduction of Testing Automation at Accountor HR Solutions Oy

Master's thesis 76 pages, appendices 5 pages
April 2019

This thesis was done for Accountor HR Solutions Oy, a company which specializes in human resources and payroll administeration solutions. The company has grown fast and during the last few years expanded their product portfolio. As versions are being updated in accelerating cycles, and the product portfolio keeps expanding and branching out, manual testing alone is no longer enough to guarantee quality in due time. This lead to the need for introduction of testing automation in Research and Development projects.

The objective of this thesis was to develop and enchance the quality and development process of the Mepco product family and ensure the operational reliability between different versions. The purpose was to introduce automated testing, create instructions and operating models for the cascading of testing and test the created instructions and recommendations in practice. The method of research used was operational research. As the base for information for this thesis were related professional literature, surveys about the company's situation, needs and development process, and the personal experience of the author on automated testing and of the company's practices and needs.

As the result of this thesis a readiness for automated testing was achieved in the target organization. The Proof of Concept demo was put under a feasibility study in the company and instructions were made. As part of test data management, the application of the General Data Protection Regulation (EU 2016/679) regarding the test data had been inspected and to assess the need for data protection and the level of the protection needed, instructions were created. The challenges detected in the introduction and well-tried practices were viewed from the aspect of change management.

As continuation it is suggested that automated testing should be cascaded more extensively. Shared responsibility of product development and the quality of testing, means that as a part of the cascading, the skills and proficiencies needed in the testing are trained to all staff in Research and Development, and not explicitly to testers. Regarding resources, it is recommended to set the automated testing role into the process, so the appropiate resources for automation can be reserved.

Key words: test automation, test data management, anonymization, pseudonymization

SISÄLLYS

LYHENTEET JA TERMIT

Anonyymi tieto	Yksittäinen havainto (henkilö) ei ole kohtuullisin keinoin tunnistettavissa annettujen tietojen perusteella tai tietoja muihin tietoihin yhdistämällä. Anonymisoinnilla kuvataan niitä moninaisia tapoja ja välineitä, joilla anonyymi tieto saavutetaan.
Attribuutti	Fakta, kuvaava termi. Tässä dokumentissa viitataan Master Datan ominaisuuksiin jotka ovat aina yhteisiä, toimintojen kesken jaettavia attribuutteja.
Avainsana	(keyword) Tässä dokumentissa viittaa avainsanapohjaiseen testaukseen ja testikirjastoihin, joissa avainsanojen taakse on kytketty toiminnallisuuksia
BDD	(Behaviour Driven Development) Käyttäytymisläheinen kehittämistapa, jossa kehittämisen lähtökohtana on tarkastella toiminnallisuutta se käyttäjätarinoiden, eli skenaarioiden pohjalta.
Bugi	Vika on ohjelmiston tai dokumentaation poikkeama, joka estää ohjelmaa toimimasta oikein aiheuttaen häiriön. Testauksen tavoitteena on löytää kaikki merkittävät bugit.
De-anonymisointi	Anonyymiksi luokitellun tiedon saattaminen tunnisteelliseksi yhdistämällä tietoja muista lähteistä saataviin tietoihin. Jos anonyymi aineisto onnistutaan de-anonymisoimaan, aineisto joko ei ole ollut alun perin anonyymi tai käytettävissä oleva teknologia on kehittynyt tai henkilöistä on saatavissa muualta aiempaa enemmän tietoja. Anonyymisoinnin kestävyyttä tuleekin arvioida määräajoin (niin sanottu jäännösriskin arviointi).
De-identifiointi	(de-identification) Tunnisteellisten tietojen poistaminen tai muokkaaminen yksittäisen havainnon tunnistamisen ehkäisemiseksi datatiedoston sisällä. Usein de-identifioinnilla tarkoitetaan suorien tunnisteiden poistamista tai peittämistä.
DevOps	Toimintamalli, jossa automatisoidaan ohjelmistokehityksen prosessi aina käännöksestä testeihin ja ylläpidosta operatiiviseen toimintaan kuten palvelun julkaisuun.

Entiteetti	Asia, tässä dokumentissa viitataan Master Data yhteen kuvattavaan osaan, esimerkiksi toimipaikka, kontaktihenkilö, tai työsuhde.
Häiriö	Häiriötä käytetään tässä dokumentissa kuvaamaan tilannetta, jossa ohjelma toimii odottamattomalla tavalla. Häiriö ei välttämättä ole bugi sovelluksessa, vaan sen voi aiheuttaa myös laitteisto, kolmannen osapuolen sovellukset, huonepöly tai mikä tahansa muu joka haittaa sovelluksen käyttöä.
Integraatiotestaus	Varmennetaan yhden tai useamman rajapinnan keskustelua järjestelmässä, tyypillisesti perustuu järjestelmän arkkitehtuuriin.
Jatkuva integraatio	(CI, Continuous Integration) Prosessi jossa ohjelmistoa käännetään ja integroidaan jatkuvasti automatisoidusti.
Jatkuva testaus	Osa jatkuvaa integraatiota, viittaa testien ajamiseen jatkuvasti
K-anonymiteetti	Anonymisoinnin arvionnin työkalu, jolla voidaan asettaa vaatimus sille kuinka uniikki henkilöön liittyvä tieto saa olla. K-anonymiteetti 3 tarkoittaisi, että tunnisteiltaan samanlaisia henkilöitä tulisi olla ainakin 3 jotta henkilöä ei voida tunnistaa tunnistetietojen perusteella.
Ketterät menetelmät	(Agile Methods) esimerkiksi Scrum, Lean, Extreme Programming (XP), tässä dokumentissa viitataan scrumiin joka valittiin menetelmäksi.
Kontti	Docker-kuva on eräänlainen virtuaalinen ympäristö jossa on sovelluksen perustiedot ja vaatimat sovellukset valmiiksi asennettuna.
Käyttötapauslähtöinen kehitys	(Behaviour Driven Development, BDD) Käyttötapauslähtöinen kehitys tarkoittaa kehitystä, jossa toimintoja tarkastellaan siihen liittyvien käyttötapausten pohjalta.
Lasilaatikkotestaus	(White Box Testing) Testausta, jossa testitapausten suunnittelussa ja toteuttamisessa käytetään hyväksi tietoa ohjelman toteutusta, eli käytännössä testin suunnittelussa tulee olla käytössä ohjelman tai sen osan lähdekoodi ja toiminnallinen määrittely.

L-diversiteetti	Anonymisoinnin arvioinnin työkalu, joka kuvaa kuinka paljon vaihtelua arvojoukossa täytyy olla. Esimerkiksi l-diversiteetti 2 tarkoittaa että yhden ryhmä tiedoissa täytyy olla vähintään kahdenlaisia arvoja jos kyseessä on arkaluontoiseksi luokiteltu tieto.
Legacy-projekti	Projekti, jonka lähdekoodia ei enää kehitetä eikä tueta.
Master Data	Organisaation ydinliiketoimintatietoa joka on luonteeltaan pitkäikäistä ja hitaasti muuttuvaa, esimerkiksi tuotetiedot, organisaatiotiedot, tai henkilörekisteri.
Metadata	Tietoa tiedosta, eli kuvaa entiteettien, relaatioiden ja attribuuttien säännöt ja asetukset.
Minimointi	Minimointi on suojatoimi, jolla pyritään vähentämään tunnisteellisten aineistojen käsittelyyn sisältyviä riskejä. Henkilötietojen käsittelyn minimointi tarkoittaa käyttötarkoituksen kannalta ei-välttämättömien henkilötietojen hävittämistä. Esimerkiksi henkilötunnus, nimitiedot, osoitteet ja muut tarpeettomat tunnisteet hävitetään heti, kun ne eivät ole enää välttämättömiä käyttötarkoitukselle.
Mustalaatikkotestaus	(Black Box Testing) Mustalaatikkomenetelmällä ohjelmaa testataan ilman tietoa ohjelman rakenteesta, lähdekoodista tai algoritmista, jonka pohjalta ohjelma on toteutettu.
Mock	Testikomponentti joka simuloi varsinaisen järjestelmän osia tai niiden toimintoja.
TDD	(Test Driven Development) Testauslähtöinen ohjelmointitapa on menetelmä, jonka pääperiaatteena on, että kehittäjät kirjoittavat matalan tason yksikkötestejä ennen ohjelmayksikköjen toteutusta.
Proof of concept-projekti	Esiprojekti tai vaihe, jossa tutkitaan mahdollisen toimintatavan tai sovelluksen soveltuvuutta käyttöön. Tavoitteena tuottaa varmuus siitä sopiiko ratkaisu käyttöön vai ei.
Pseudonyymi tieto	Yksittäinen havainto ei ole tunnistettavissa pseudonymisoitujen tietojen perusteella ilman ulkopuolisia lisätietoja. Pseudonymisointi on aineiston tunnisteellisten tietojen poistamista tai korvaamista peitetiedolla tai koodeilla, jotka prosessin jäl-

	keen säilytetään erillään aineistosta ja suojataan organisatorisesti ja teknisesti. Aineisto on pseudonyymi niin pitkään kuin erillään oleva tieto tuhotaan.
Refaktorointi	Ohjelman refaktoroinnilla tarkoitetaan koodin uudelleenkirjoittamista.
Referenssidata	Muita datoja luokitteleva tieto, joka on määrämuotoista ja rajattua, esimerkiksi ISOn maalista.
Regressiotestaus	Testataan uudelleen jo toimivaksi varmennettua ominaisuutta tai toimintoa sen varmistamiseksi, ettei vika ilmene uudelleen sovelluksen päivityksen yhteydessä.
Relaatio	Yhteys tai suhde asioiden välillä. Tässä dokumentissa viitataan Master Datan relaatioihin joissa tarkastellaan entiteettien välisiä suhteita, esimerkiksi toimipaikan ja työsuhteen välillä.
RUP	Yhtenäinen prosessi (Rational Unified Process) on iteratiivista ohjelmistokehitystä, joka perustuu siihen, että jokaisessa neljässä vaiheessa (aloitus, tarkennus, rakentaminen ja siirtyminen), käsitellään sekä mallinnusta, vaatimusmäärittelyä, analysointia, testausta että käyttöönottoa, joskin niiden painotus vaihtelee kehitysvaiheen mukaisesti.
Savutestaus	Koostuu regressiotesteistä, jotka varmistavat järjestelmän perustoiminnallisuutta ja todentavat että tietyn perustason vakaus järjestelmässä on kunnossa.
Scrum	Ketterän ohjelmistokehityksen projektihallinan viitekehys joka perustuu siihen että yksittäinen kehitysjakso on lyhyt, jolloin ominaisuus kehittyy iteratiivisesti.
Skenaario	Käyttäjätarina, tässä dokumentissa viitataan BDD mallin mukaiseen käyttäytymiskuvaukseen.
Stub	Sijaiskomponentti integraatiotestauksessa, korvaa jonkin toiminnon tai osan.
Suora tunniste	Suora eli välitön tunniste on sellainen henkilö tieto josta käsiteltävä henkilö voidaan suoraan päätellä, esimerkiksi henkilötunnus tai koko nimi.
Validointi	(Validation) Validoinnilla eli kelpoistamisella varmennetaan, että toteutettava järjestelmä vastaa loppukäyttäjän tarpeita.

Verifiointi	(Verification) Verifioinnilla eli todentamisella, tarkoitetaan ohjelman oikeellisuuden ja oikean toiminnan todentamista.

1 JOHDANTO

Digitalisaatio kulkee käsi kädessä testausautomaatioon kohdistuvan kasvavan kiinnostuksen kanssa. Vaikka testausautomaatiota yhä perustellaan myös taloudellisilla näkökohdilla, ketterän kehityksen myötä halutaan, että tuotteella on jatkuva integraatio käytössä ja jatkuva julkaisu edellyttää myös jatkuvan testauksen. Jatkuva integraatio tarkoittaa automatisoitua prosessia, jossa sovellusta käännetään ja integroidaan jatkuvasti. Täydellisessä maailmassa virheitä ei synny eikä siksi testauksellekaan ole tarvetta. Täydellistä ja virheetöntä koodia ei kuitenkaan ole, vaan ohjelmistokehittäjät käyttävät jopa puolet ajastaan virheiden korjaamiseen. Endresin ja Rombachin (2003, 125) mukaan ei ole väliä sillä, kuinka huolellista ohjelmiston suunnittelu ja kehitys on ollut: testausta ei tulisi koskaan jättää tekemättä. Ohjelmistojen testaus on olennainen osa sovelluksen kehitystyötä, sillä sen rooli on kehitystyön tuotoksen verifiointi ja virheiden löytäminen (Endres & Rombach 2003, 123).

Sovellusten siirtyessä pilveen sovellusten päivitystiheys on kasvanut nopeasti ja manuaalinen testaus ei anna riittävää varmuutta sovelluksen toimintavarmuudesta kyllin nopeassa vasteajassa, vaan muodostuu pullonkaulaksi. Toisaalta kasvava datamäärä ja käyttötapausten määrä tekevät mahdottomaksi testata kaikkea manuaalisesti. Tällöin automaatiotestaus voi tarjota paitsi ajantasaista tietoa tuotteen laadusta ja suorituskyvystä, myös keventää manuaalisen testauksen testaustaakkaa. Toimiva testausprosessi mahdollistaa nopean ja joustavan reagoinnin ongelmatilanteissa, kun virheen paikka voidaan rajata ja kartoittaa nopeasti.

Testausautomaation tarkoitukseksi nostetaan yleensä laadun parannukset ja sen tuomat kustannussäästöt. Sijoitetun pääoman tuotto muodostuu testien jatkuvista toistoista, jolloin henkilöstöresursseja pystytään vapauttamaan toistuvista ja varmistavista testeistä. Automatisoidut testit voidaan ajaa useammin, nopeammin ja suuremmilla massoilla. Testausautomaatio on riskinhallinnan työkalu, sillä se antaa nopeaa tietoa tuotteen tilasta. Koska testit ajetaan aina samalla tavalla, tulokset ovat verrattavissa aiempien ajokertojen tuloksiin. (Fewster & Graham 1999, 346-347.)

Automaatiotestauksen suunnittelu on osa kehittämisprosessien uudistus- ja parannusprojektia. Automaatiotestauksen käyttöönoton suunnittelua varten koottiin konsernitasoinen

automaatiotestausryhmä. Opinnäytetyön tekijän rooli on viedä automaatiotestausta eteenpäin omassa yrityksessään Accountor HR Solutions Oy:ssa. Kohdeorganisaatiossa manuaalisen testauksen toiminta on prosesseiltaan vakiintunut, mutta automatisoitua testausta ei lähtötilanteessa tehdä.

Opinnäytetyön tavoite oli kehittää ja parantaa Mepco-tuoteperheen laatua ja kehitysprosessia sekä varmistaa toimintavarmuutta eri versioiden välillä. Tarkoitus oli ottaa käyttöön jatkuva testaus, luoda ohjeistukset ja toimintamallit testauksen jalkautukseen sekä testata luodut ohjeistukset ja suositukset käytännössä. Tutkimusmenetelmänä oli toimintatutkimus. Opinnäytteessä testausautomaation käyttöönottoa käydään läpi muutosjohtamisen tarpeen ja siinä ilmenneiden haasteiden kautta. Prosessia tarkastellaan testauksen hallinnan, työkalujen ja testidatan näkökulmasta. Tavoitteena on opinnäytteen aikana saavuttaa automaatiotestauksen valmius yrityksessä.

Automaatiotestauksen käyttöönotossa lähdettiin käytössä olevien työkalujen ja olemassa olevien käytäntöjen kartoituksella. Testauksen tarvearviointia ja tarpeiden priorisointia käydään läpi luvussa 2, jossa käydään läpi myös regressiotestausta. Regressiotestauksella tarkoitetaan testausta, jonka päämäärä on varmistaa, että aiemmin toteutetut toiminnallisuudet toimivat virheettömästi uusien muutosten jälkeen. (Kasurinen 2013, 68.)

Työkaluille tehtiin soveltuvuusselvitys (proof of concept). Käyttöön valituille työkaluille tehtiin organisaatiokohtaiset säännöt käytöstä ja yrityskohtaisesti tiimin vastuuhenkilö kävi koulutuksia niiden käyttöönottoa varten. Käyttöönottoa käydään läpi luvussa 3.

Projektin aikana tuotekehitysprosessissa tehtiin suuria muutoksia kohti ketterää ohjelmistokehitystä. Toimintatapojen muutosta käydään läpi opinnäytteessä testauksen näkökulmasta, painottuen kuitenkin automaatiotestauksen näkökulmaan. Lähestymistavaksi valittiin käyttötapauslähtöinen kehittäminen (behaviour-driven development, BDD), jonka keinoin pyritään paitsi parantamaan sidosryhmien kommunikaatiota, aikaistamaan integraatiotestien toteutusta ja suunnittelua sekä tuomaan testausautomaatio osaksi toimintatapoja. Testien suoritusta ja valittuja työkaluja käydään läpi luvussa 4, Automatisoitujen testien suorittaminen.

Opinnäytteen raportointiosuus rajautuu käyttöönottovaiheeseen ja datan hallintaan. Datan hallintaa ja etenkin testidatan käsittelyä käydään tarkemmin läpi luvussa 5. Opinnäytteessä otetaan kantaa manuaaliseen testaukseen vain siltä osin kuin se vaikuttaa automaatiotestaukseen. Tästä syystä manuaalisen testauksen piiriin kuuluvat testauskohteet, kuten käytettävyys, rajataan pois.

Projektissa keskitytään yksikkö- ja hyväksyntätestaukseen sekä sivutaan järjestelmätestausta, koska käyttöönotossa ensimmäisiksi automatisointikohteiksi valittiin yksikkö- ja hyväksyntätestaus. Tietoturvatestaus rajattiin ulkopuolelle käyttöönottovaiheessa.

2 AUTOMATISOITU TESTAUS

2.1 Testausmenetelmät

Testausmenetelmät ovat jaettavissa staattiseen ja dynaamiseen testaukseen. Staattisella testaamisella tarkoitetaan sitä, että sovellusta ei suoranaisesti käytetä, vaan sitä tarkastellaan koodikatselmoinnin, arkkitehtuurisuunnittelun tai analysaattoreiden avulla. Staattisessa testauksessa varmistetaan ratkaisun logiikan olevan kunnossa. (Kasurinen 2013, 65.)

Staattinen testaus rajataan opinnäytteen ulkopuolelle, mutta siihen viitataan, kun tarkastellaan tuotteen laadun varmistusta. Esimerkkinä staattisen testauksen roolista automaatiotestauksessa on koodin tai testien katselmointivaihe, jossa käydään läpi, noudattaako ratkaisu hyvää koodaustapaa ja onko toteutuksessa riskejä.

Dynaamisessa testauksessa sovellus on käytössä ja sen reaktioita syötteisiin tarkkaillaan. Suurin osa automatisoidusta testauksesta on dynaamista testausta. Staattisen testauksen etu on sen hinta – kun ongelmat havaitaan jo ennen toteutuksen alkua, ne ovat edullisempia korjata kuin siinä vaiheessa, kun toteutus on jo pitkällä. (Kasurinen 2013, 65.)

Jotta sovelluksella on valmius jatkuvaan julkaisuun, on sen kyettävä jatkuvaan testaukseen. Testauksen on oltava osa prosessia jo tuotannon alussa, jotta virheistä ja odottamattomasta toiminnasta toivutaan nopeasti. Automaatiotestausstrategian tulee kattaa kaikki tuotteen tasot. (Banerjee 2018.)

Mustalaatikkotestaus ei ota kantaa siihen, mitä sovelluksen sisällä tapahtuu. Mustalaatikkotestauksessa varmennetaan, että oikeat tiedot palautuvat annetuilla syötteillä. Menetelmän etu on sen yksinkertaisuus ja käytettävyys melkein missä tahansa työvaiheessa, jossa toiminnallisuutta suoritetaan. Mustalaatikkotestauksen suurin haaste on se, että testaaja ei tiedä onko oikea tulos sattumaa. Ratkaisun riskit tai huonot koodausratkaisut eivät tule ilmi, joten katselmoinnin merkitys korostuu. Mustalaatikkotestaus on kuitenkin käytettävissä vain silloin kun annetut syötteet ja tulokset pystyvät todentamaan toiminnallisuuden toimivuuden. (Kasurinen 2013, 66.)

Lasilaatikkolähestymistavassa ohjelmalle annettujen syötteiden käsittelyä myös tarkastellaan. Testaaja pystyy jäljittämään mistä virhe aiheutui, ja pystyy seuraamaan koko käsittelyn loogisella tasolla, joten hän voi varmistua, että tulokset ovat oikeita oikeista syistä. Jotta testaaja pystyisi tekemään lasilaatikkotestausta, hänen pitää tuntea järjestelmä niin hyvin, että voi tarkastella sitä lähdekooditasolla. Sen lisäksi testaajan pitää pystyä ymmärtämään ohjelmointityö ja järjestelmän toimintalogiikka niin hyvin, että pystyy varmuudella arvioimaan koodin toteutusta. Lähestymistavan heikkous on se, että se ei ota kantaa vaatimusmäärittelyn heikkouteen ja vaatii syvällistä osaamista testaajalta. (Kasurinen 2013, 68.)

Molemmista lähestymistavoista voidaan ottaa parhaat puolet, mitä kutsutaan harmaan laatikon testaukseksi. Lähestymistavassa käytetään vaatimusmäärittelyssä tehtyjä mustan laatikon testitapauksia läpi ja hyödynnetään lasilaatikkotestien tapaan järjestelmän toteutusta ja luodaan näiden perusteella testitapaukset. Tämä menetelmä toimii erityisen hyvin tilanteisiin, joissa mukana on oma sovellus, jota voidaan tarkastella lasilaatikkotasolla sekä kolmannen osapuolen järjestelmä, josta tunnetaan vain rajapinta. (Kasurinen 2013, 68.)

2.2 Automaatiotestauksen hyödyt

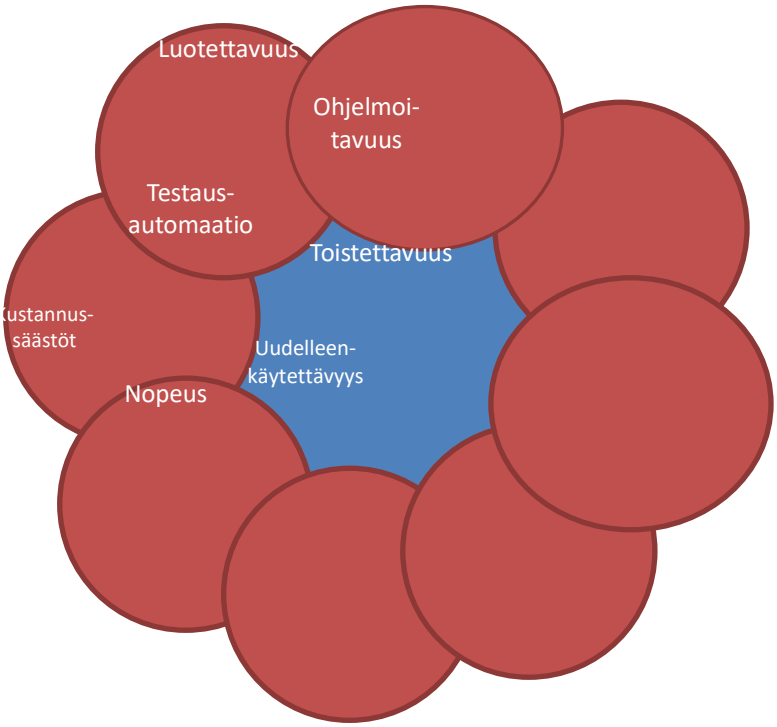
Automaatiotestauksessa yhdistyy kolme teemaa: 1) tekoäly ja koneiden oppiminen, 2) automatisointi ja 3) jatkuva integraatio, jolla käsitetään toiminnot jatkuvasta testauksesta aina jatkuvaan toimitukseen. Digitalisoituminen ja pilvipalvelut ovat muodostaneet tarpeen julkaista sovellusta nopeassa syklissä. Kun sovellusta julkaistaan automaattisesti tietyllä aikavälillä, puhutaan jatkuvan julkaisun organisaatiosta. Perinteisessä julkaisusyklissä julkaistaan uusia sovellusversioita muutamia kertoja vuodessa, kun taas jatkuvan julkaisun organisaatio saattaa julkaista uusia versioita useita kertoja minuutissa. (Banerjee 2018.)

Automatisoidulla testauksella halutaan vähentää manuaalisen työn tarvetta, nopeuttaa testausta ja parantaa tuotteen laatua. Parhaiten automatisoitaviksi sopivat testit joita 1) toistetaan usein, 2) ovat määrämuotoisia syötteiltään ja vastauksiltaan, ja 3) ovat tarpeen järjestelmän toimintakunnon arvioimiseksi. Automaatiotestaus soveltuu hyvin regressiotestaukseen, eli sen varmistamiseen, että toimivaksi todetut osat toimivat yhä uusien päivitysten ja muutosten jälkeen. (Kasurinen 2013, 77-78.)

Kustannustehokkuutta arvioitaessa huomiota kiinnitetään testin elinkaarikustannuksiin. Yksittäisen testitapauksen tekeminen automatisoituna on paljon hitaampaa ja kalliimpaa kuin manuaalisen testin tekeminen, projektin aikana seuratuista tiketeissä havaittiin, että automatisointi saattoi viedä kymmenen kertaa enemmän aikaa. Mutta automatisoidun testin toistaminen on lähes ilmaista. Automatisoitujen testien monitorointi ja ylläpito vaativat kuitenkin resursseja, joten automatisointi toimii parhaiten tilanteisiin, joissa testejä muutetaan harvoin tai ne vaativat vain vähäisiä muutoksia ylläpitotöinä. (Kasurinen 2013, 79.)

Automatisoitujen testien kustannustehokkuus syntyy toistojen määrästä: koska ne toteutuvat aina samoin, ne ovat tarkkoja ja nopeita. Tarkkuuden ja nopeuden ansiosta testit mahdollistavat muiden mittarien, kuten järjestelmän suorituskyvyn kehittymisen, seurannan. Kun tutkivien testaajien resurssit voidaan vapauttaa tärkeämpiin testeihin, sovelluksesta saadaan enemmän ja syvällisempää tietoa. (Kasurinen 2013, 78.)

Kuviossa 1 automaatiotestauksen edellä mainitut hyödyt kootaan yhteen. Hyödyt ovat toisistaan riippuvaisia. Ohjelmoitavuus tuo testattavuutta, testattavuus uudelleenkäytettävyyttä. Uudelleenkäytettävät testit ovat nopeita toistaa, ja toistot tuovat kustannussäästöä, joka kiteytyy sijoitetun pääoman tuotossa. Luotettavuus ja kustannustehokas testaus parantavat tuotteen laatua.



KUVIO 1. Automaatiotestauksen hyödyt (Heimola 2017, muokattu)

Automaatiotestauksen suurimmat hyödyt saavutetaan, koska se vaatii mahdollisimman vähän manuaalista työtä tuottaen kuitenkin hyvää tulosta. Muita syitä testien automatisointiin on, että niillä voidaan saada helposti aikaan tilanteita, joita manuaalisesti olisi vaikea tai kallis saada aikaan, esimerkiksi voidaan luoda tuhansia käyttäjiä käyttämään sovellusta yhtä aikaa.

Kuviossa 2 on taulukoitu, paljonko aikaa rutiinitehtävän muuttamiseen tehokkaammaksi voi maksimissaan käyttää, jotta se säästää aikaa enemmän kuin tekemiseen menee aikaa viidessä vuodessa. Taulukossa korostuu toistokertojen vaikutus: mitä useammin testiä toistetaan, sitä suurempi säästö automatisoinnilla saavutetaan.

Kauanko toiminnon tekemiseen voi käyttää aikaa ennen kuin kuluttaa enemmän kuin säästää (viidessä vuodessa)							
		Toiminnon toistokertojen määrä					
		50/pvä	5/pvä	Päivittäin	Viikottain	Kuukausittain	Vuosittain
Paljonko aikaa säästyy	1 s	1 pvä	2 h	30 min	4 min	1 min	5s
	5 s	5 pvä	12 h	2 h	21 min	5 min	25 s
	30 s	4 vko	3 pvä	12 h	2 h	30 min	2 min
	1 min	8 vko	6 pvä	1 pvä	4 h	1 h	5 min
	5 min	9 kk	4 vko	6 pvä	21 h	5 h	25 min
	30 min		6 kk	5 vko	5 pvä	1 pvä	2 h
	1 h		10 kk	2 kk	10 pvä	2 pvä	5 h
	6 h				2 kk	2 vko	1 pvä
	1 pvä					8 vko	5 pvä

KUVIO 2. Automatisoinnin kannattavuuden arviointi

Kustannussäästöt nostetaan usein automaatiotestauksen käynnistyksessä esille, mutta yhä useammalle organisaatiolle automaatiotestaus on välttämättömyys matkalla kohti jatkuvaa julkaisua. Automaatiota voidaan testien ajamisen ohella hyödyntää myös muilla testaamisen osa-alueilla, kuten esimerkiksi tulosten analysoinnissa, virheiden monitoroinnissa ja raporttien laatimisessa (Myers, Sandler & Badgett 2012, 38). Tämä tarkoittaa uuden testattavan sovellusversion saapuessa automaatiotestit menisivät automaattisesti suoritukseen, ja viimeisen testin suorittamisen jälkeen tulokset ja kattavuusraportit generoituisivat automaattisesti.

Käyttöönottoprojektissa päädyttiin käyttäytymisläheiseen kehittämistapaan, jossa kehitystä ohjaa käyttäjätarinat eli käyttäytymistä kuvaavat skenaariot. Kehittämistavan selkein etu on selkeys, sillä skenaariot ovat selkokielisiä kuvauksia siitä, miten tuotteen odotetaan käyttäytyvän eri tilanteissa. Toimintamalli on suunniteltu nopeuttamaan kehitysprosessia. Skenaariot sisältävät vaatimukset, hyväksyntäkriteerit, testitapaukset ja testausskriptit. Kaiken lisäksi niitä voidaan hyödyntää virheiden raportoinnissa ja uudelleen toistamisessa. Näin ne ovat elävä dokumentaatio tuotteesta.

Lähestymistavassa kiteytyy testausautomaation hyödyt, sillä skenaariot ovat uudelleen käytettäviä ja ohjaavat kehittämään uudelleenkäytettävää koodia. Skenaarioiden kirjoitus on nopeampaa ja helpompaa mitä enemmän niitä on jo tehty. Skenaariot mahdollistavat helposti erilaisten variaatioiden testaamisen. Toisaalta selkokielistä skenaariota on helppo päivittää, kun tuote muuttuu.

2.3 Testausautomaation haasteet

Useimmiten testiautomaation implementointi ei ole vaikeaa - vaikeaa on tehdä se tehokkaasti ja hyvin. Automaatio on helppo ottaa mukaan alkuvaiheessa projektia, mutta vaikeutuu myöhemmin. Legacy-projektiin testiautomaation lisääminen voi olla erittäin haasteellista ja vaatia refaktorointia, jotta ympäristö on testattavissa. Suurin haaste useimmiten on, että tiimiltä puuttuu tietotaito tehdä testaus hyvin.

Automaatiotestauksen haasteet lähtevät testauskohteen ja tarpeen ymmärtämisestä. Esimerkiksi regressiotestit ajetaan aina samalla tavalla, joten automaatio sopii niihin, kun taas käytettävyyttä ei automatisoinnilla voi testata lainkaan. Regressiotestien tarkoitus on varmistaa, että järjestelmä toimii edelleen. Jos regressiotesteissä löytyy paljon virheitä ja usein, on syytä tarkastaa tuotekehityksen tila. (Gharhai 2015.)

Automaatiotestauksen ydin on tunnistaa avainskenaariot. Varsin usein ongelma tuotannossa johtuu siitä, että kehityksen aikana kukaan ei ole tullut ajatelleeksi skenaariota, jossa ongelma nousee (Gharhai 2015). Jos avainskenaarioiden tunnistamiselle ja määrittelylle ei ole annettu riittävästi huomiota, ei ole merkitystä, kuinka monta kertaa testit on ajettu: virheitä pääsee läpi.

Automaatiotestit tulee tehdä oikeaan aikaan, oikeaan tasoon ja oikealla työkalulla. Kun käyttäjätarinasta automatisoidaan hyväksyntäkriteereitä, on tunnettava sovelluksen rakenne. Useat web-sovellukset jakautuvat selkeästi käyttöliittymäsovelluksen ja logiikkasovelluksen välille. Käyttöliittymätestit tarjoavat käyttöliittymäkehittäjälle nopeasti palautetta siitä, miten UI-komponentit toimivat. Logiikkakerros olisi hyvä testata jo rajapinnan kautta. (Gharhai 2015.)

Testien automatisointi itsessään ei ratkaise ongelmia ja tee tuotteesta laadukkaampaa. Se on laadun varmistamista, mutta laadun parantuminen riippuu tavoista, joilla havaintoihin reagoidaan. Testien automatisointi ei ratkaise prosessiongelmia, joskin tieto virheistä tulee nopeammin kuin manuaalisesti testattuna. Automatisointi ei yleensä paljasta paljoa uusia havaintoja, koska usein automatisointi tehdään vasta kun ominaisuuden jo tiedetään toimivan. (Banerjee 2018.)

Menestyksekkään testausautomaation laatiminen, eli testiympäristöjen luonti ja konfigurointi tehokkaasti vaatii paljon työtä ja osaamista. Osaamista tarvitaan testialustan luomiseen ja hallitsemiseen, sovellusten kääntöön ja prosessinhallintaan, teknisten haasteiden ratkaisemiseen ja yhdistämään tuoteomistajien, kehittäjien ja testaajien osaaminen laadun varmentamiseksi. (Banerjee 2018.)

Organisaation automatisoinnin kehitys projektin aikana noudattaa pitkälti ohjelmistotestauksen aaltoja. Ensimmäinen aalto on manuaalinen testaus ja valmistajakohtaiset työkalut. Manuaalisen testauksen suurin etu on hyvä käyttäjäkokemus ja siksi se on osa laadunvarmistusta. Manuaalinen testaus on kuitenkin tehotonta ja hidasta usein toistettavissa tehtävissä, mistä sai alkunsa toinen aalto, testauksen automatisointi. Automatisoinnin haaste on kuitenkin, että se tutkii vain sen mitä se on ohjelmoitu tutkimaan. Määrittelyiden täytyy olla tarkat, jotta testi osaisi testata kaikkea. Automatisointi tuottaa myös paljon dataa minkä vuoksi monitorointi ja hallinnointi voi olla paljon resursseja vaativaa. (Rowe 2014.)

Kolmannessa aallossa tunnusomaista on työkalumaisuus, eli tehdään sovellus, joka kirjoittaa testit. Tässä työkalulle annetaan vaatimukset ja se tuottaa testitapaukset automaattisesti. Mallipohjainen testaus on yksi työkalutestauksen muoto. Testaustapa ei ole vielä yleisesti käytössä, koska vaikka mallilla on helppo generoida paljon testitapauksia, tarkastuksien säännöt ovat usein vaikeampia määritellä ja vaativat ihmisen arvioimaan ja

läpikäymään ratkaisut. Automatisointi ei osaa päätellä testin löytämän virheen prioriteettia, tai mikä sen todellinen ilmenemistodennäköisyys on. Kolmannessa aallossa automatisointia viedään tutkivaan suuntaan, ja automatisoinnilla voidaan löytää uusia havaintoja, mikä erottaa sen toisesta aallosta. (Rowe 2014.)

Oikeiden työkalujen valinta on tärkeää, sillä automatisointi aikaisessa vaiheessa on joillakin työkaluilla vaikeaa, ja suuri osa hyödyistä jää saavuttamatta, jos automatisointi tehdään vasta, kun ominaisuus tai tuote on valmis. Projektissa ongelma ratkaistaan testin määrittelykerroksella, joka voidaan tehdä ympäristöstä riippumatta.

Toimiva kommunikaatio automaatiotestaajien, tuoteasiantuntijoiden, testaajien, kehittäjien ja sidosryhmien välillä on ketterän kehittämisen perusedellytyksiä. Mikäli muutoksia määrittelyyn ja vaatimuksiin tulee koko ajan, mutta tieto muutoksista kulkee hitaasti, sekä datan että testien eheys vaarantuu. Automaatiotestit testaavat vain ne skenaariot, joita se on määritelty testaamaan. (Smart 2015, 16.)

Testauksen automatisoinnin kannalta vaikeita asioita ovat testit, joissa 1) tarvitaan eri teknologioita yksittäisen asian testaamiseen, b) data on erittäin dynaamista, jolloin myös elementtien ID:t voivat vaihtua joka kerta testiä ajaessa, c) tapahtumia täytyy odotella, d) virhevaroituksia nousee odottamattomasti ja usealla eri tavalla, e) työnkulku on monimutkainen, jolloin yhden testin hajoaminen vaikuttaa seuraaviin askeliin ja järjestys on tärkeässä roolissa ja f) testauselementit ovat vaikeita toteuttaa, esimerkiksi mobiilitestauksessa puhelimen soitto kesken testin suorittamisen, tai vähäinen testilaitteen vähäinen virta testin suorituksen aikana.

Testauskohteen vaikeus pelkästään ei ole syy olla automatisoimatta testiä, mutta kasvanut resurssien tarve on huomioitava testin kannattavuutta arvioitaessa. Sellaisia testejä ei pitäisi automatisoida lainkaan, jotka ajetaan vain kerran manuaalisesti, joista ei ole määrittelyä ja tietoa, mitä tuloksien pitäisi olla. Koska tällöin hylkäyskriteerit eivät ole selvillä eikä testi välttämättä testaa oikeita asioita. (Ghahrai 2015.)

Tietyt ominaisuudet vastustavat automatisointia, mistä tyypillinen esimerkki on CAPTCHA:t web-lomakkeilla. Sen sijaan, että käsittely yritetään automatisoida, se on parempi ottaa pois käytöstä testiympäristöstä. Epävakaille ja suuren muutoksen alla oleville ominaisuuksille ei myöskään kannata tehdä testejä ennen kuin tila on vakiintunut.

Poikkeuksena yksittäisten moduulien yksikkötestit, jotka todennäköisesti toimivat vaikka kokonaisuuteen tehdään muutoksia ja joiden tekeminen toimii kehityksen tukena siinä missä toiminnan varmistajanakin.

Automaatiotestaus on nähtävä ylläpidettävänä kokonaisuutena. Yksi tyypillinen virhe on tehdä ja ajaa testit kokonaan käyttöliittymän kautta. Keskeiset haasteet lähestymistavassa ovat se, että nauhoitettujen testien ajo käyttöliittymän kautta on hidasta ja testien määrän kasvaessa minuutit kasvavat nopeasti tunneiksi. Ja jos testit ovat riippuvaisia toisistaan, virhe aikaisessa vaiheessa voi estää useiden testien ajon. Pahimmassa tapauksessa testaajat ylläpitävät ja etsivät vääriä virheitä. Parhaisiin tuloksiin päästään kiinnittämällä huomioita useisiin testaustasoihin. Yksikkötestauksessa kehittämisvaiheessa varmistetaan vaatimusten toteutuminen yksittäisillä sovelluksen osilla. Integraatiotesteillä varmistetaan tiedon oikea käsittely useiden rajapintojen kautta. Käyttöliittymätestit on syytä nähdä hyväksyntätesteinä, joilla varmistetaan avainskenaarioiden oikea toiminta. Yhdessä testaustasojen testit mahdollistavat ongelmien aiemman havaitsemisen, testien nopeamman suorituksen ja helpottavat ongelman ilmetessä vian. (Gharhai 2015.)

Kääntö-testaus-julkaisu -ketjun huomiotta jättäminen on virhe testausautomaation aloittamisessa. Vaikka testit voidaan tehdä ulkoisesti skriptaamalla ja saavuttaa näin tuloksia ja saada illuusio testauksen nopeudesta, suuri osa automaatiotestauksen resursseista kuluu ylläpito ja tukitöihin. Testiympäristön asennus, testien suunnittelu, strategia, testidatan luonti ja asennus ovat kaikki osa testausta. Testit, joita täytyy ajaa käsin, voivat jäädä testaamatta silloin kun niiden ajo olisi kaikkein merkittävintä. Siksi on parempi lisätä testejä jatkuvaan integraation vähitellen ja harkiten. (Gharhai 2015.)

Haasteellisissa ympäristöissä testiautomaation haasteeksi voivat nousta tekniset ongelmat, jos työkalut eivät ole yhteensopivia järjestelmän kanssa, tai työkaluissa itsessään on virheitä. Tämän vuoksi proof of concept on tärkeä käyttöönottovaihe, jossa haasteiden ratkaisu varmistetaan. Testausautomaatio vaatii sekä teknisiä taitoja että tukea hallinnolta jolloin osa automaatiotestauksen haasteista voi olla lähtöisin myös organisaatiosta itsestään.

2.4 Testauksen nykytilan arviointi

Testiautomaation käyttöönotto alkoi nykytilan arviolla. Automatisoituja testejä ei käytännössä ollut. Nykytilan arviointi on kuitenkin muutakin kuin nykyisten testien läpikäyntiä, sillä tällöin määritellään, miksi testiautomaatiota tarvitaan, ja onko tiimi valmis siihen. Löytyykö tarvittava osaaminen ja resurssit? Mitkä ongelmat automaatiotestauksella pyritään ratkaisemaan? Tukeeko nykyinen kehitysprosessi automaation käyttöönottoa ylipäätään? Täytyy myös ennakoida tulevaa, mihin suuntaan prosessit ja käytetyt teknologiat ovat kehittymässä, sillä nämä kaikki vaikuttavat myös automaatiotestaukseen.

Ongelmia lähtötilanteessa löytyi sekä resursoinnista että itse tuotteen testauksen kypsyyden arvioinnista. Testausresurssit olivat epätasaisella käytöllä, ja kuitenkin välillä ylityöllistettyjä. Päivitysversiossa testaus hidastaa julkaisutahtia, kun muutoksen lisäksi myös muuta toiminnallisuutta täytyy testata laadun varmentamiseksi. Virheiden päästessä asiakkaalle asti, ongelmat työllistävät useita henkilöitä lähtien käyttötuesta kehitykseen. Jos versiossa on virheitä paljon, version kehitysresurssit kuluvat virheiden korjaamiseen ja niistä viestimiseen, eikä uusia ominaisuuksia pystytä tuottamaan asiakkaalle tavoitellussa julkaisusyklissä.

Prosessit eivät myöskään olleet täysin valmiit automaatiotestauksen parhaan hyödyn saavuttamisen näkökulmasta. Jos testaus aloitetaan vasta muutoksen kehityskaaren lopussa, ei automaatiotestaus välttämättä tarjoa nopeaa apua, sillä testaus alkaa liian myöhäisessä vaiheessa. Tästä syystä suunnittelussa täytyy huomioida muutokset prosessiin ja miettiä roolitusta ja tarpeita uudelleen, jotta testausta tapahtuu läpi kehitysprosessin. Kehitysprosessin muutoksia käydään läpi tarkemmin luvussa 2.5 Testauksen suhde tuotekehitysprosessiin. Tulevaisuuden tahtotila on siirtyä DevOps lähestymistapaan, eli jatkuvaan toimitukseen.

Eräänä selkeänä haasteena organisaatiossa havaittiin tiimien jako suhteessa sovellusarkkitehtuuriin, sillä käytännössä samoja ominaisuuksia on toteutettuna osittain samoihin palvelintason koodeihin tukeutuen useamman tuotteen ja tätä kautta myös tiimin työalueelle. Käytännössä tämä tarkoittaa, että eri tiimien toteuttaessa melko samaa toiminnallisuutta mutta eri käyttöympäristöön, testejä saatetaan kirjoittaa samalle ominaisuudelle lisää, mikäli testien hallinta ei ole kunnossa, ja viestintätapoja ei ole mietitty. Saman asian testaus tuskin tuo lisäarvoa, jollei testauksen kohde ja tavoite ole selvillä.

Tuoteperheen laajentuessa muutamassa vuodessa yhdestä tuotteesta tuoteperheeksi, henkilöresurssit eivät riitä testauksen suorittamiseen. Selkeitä hälytysmerkkejä testauksen huonosta tilanteesta on, jos testauksesta puuttuu suunnitelmallisuus, oli syynä ajan tai motivaation puute. Tällöin testauksen tavoitteena on vain osoittaa toteutuksen toimivuus. (Haikala & Mikkonen 2011, 205-206.)

Ad hoc-testaukseksi kutsutaan testausta, jossa loppuvaiheessa kiireessä muutama ihminen tarkastaa, että kaikki näyttäisi toimivan. Tämä on yleinen toimintatapa, johon joudutaan venyvien työvaiheiden, ylioptimisten aikataulujen tai alimitoitettujen resurssien vuoksi. (Kasurinen 2013, 63.)

Kohdeorganisaation manuaalinen testaus on kyllä suunnitelmallista ja huolellista, mutta aikataulupaineiden vuoksi myös kaavamaista, ja tutkivan testauksen määrä ilmeni käyttötapauskattavuudeltaan puutteelliseksi. Vaikka luonnollisesti on tärkeää keskittää huomiota odotettuihin käyttötapauksiin, tarkoittaa tutkivan testauksen vähyys sitä, että asiakkaan määritellessä prosessin odottamattomasti, toiminta ei olekaan yhtä vakaalla pohjalla kuin aiemmin. Tästä syystä arvioitiin, että tutkivan testauksen laatu paranee, kun manuaalisen testauksen varmistavaa luonnetta voitaisiin automatisoida ja resursseja vapautuu.

Nykytilan arvioinnissa tuli esiin kehityspisteenä kehittäjien ja testaajien yhteistyö. Lähtötilanteessa työ oli jo testauksessa keskustelun alkaessa, alkuvaiheessa oli muutettu toimintaa jo niin, että kehittäjät ja testaajat istuivat samoissa versiopalavereissa, mutta näiden palaverien synergiahyödyt olivat alkuvaiheessa vähäiset.

2.5 Testauksen suhde tuotekehitysprosessiin

Jokainen yrityksessä on oma testauskulttuuri, eli oma tapa tehdä testausta. Toimintatavat ja työn vaatimusten määrittely muodostavat pohjan testauskulttuurille. Ohjelmistotestauksen keskeinen tavoite on varmistaa, että tuote toimii tarkoituksenmukaisesti, mutta tavat, joilla tavoitteisiin päästään, ja millaiset näkökulmat ja asiat korostuvat tärkeimmiksi, vaihtelevat suuresti. Käytännössä testaus on jatkuva vertailutehtävä, jonka tarkoituksena on tunnistaa, milloin toteutus poikkeaa suunnitellusta ja odotetusta.

Testaus on kokonaisuutena laajempi kokonaisuus kuin esimerkiksi ohjelmointityö tai tekninen dokumentointi ja siksi testaustyön sisältö vaihtelee paljon yritysten välillä. Testaustyö ja sen hallinnointi on kokonaisuus, jossa huomioidaan valtava määrä muuttujia kuten työkalut, ihmiset, lainsäädäntö, päätökset ja tapakulttuuri.

Kohdeorganisaatiossa manuaaliselle testaukselle on hyvin muotoutuneet prosessit ja alan parhaita asiantuntijoita sisältävät tiimit, mutta automatisoitu testaus ei ollut vakiintunut käyttöön. Ensimmäisinä toimenpiteinä tehdyssä selvityksessä haettiin vastausta sille, miksi toiminta ei ollut jalkautunut käyttöön, minkälainen motivaatio tiimeillä automaatiotestaukseen on, ja miten testauksen tarve koetaan. Tämä muodosti perustan havainnoinnille ja tietojen keruulle pitkin projektin etenemistä. Haastattelemalla ja analysoimalla eri toimissa toimivia henkilöitä haettiin vastaus sille, miksi testaamme, mitä siitä odotamme ja mitkä ovat tärkeimmät testauskohteet, joista muodostui käyttöönottoprojektin kulmakivet ja ohjenuorat.

Motivaatio testauksen tekemiselle on pohjimmiltaan riskienhallinta, sillä puutteellisen tai vajavaisen testauksen kautta päässeet bugit eli virheet ohjelman toiminnassa aiheuttavat tappioita ja tuotannonmenetystä puhumattakaan vahingoista yrityksen julkisuuskuvaan.

Suurimmaksi kynnykseksi automatisoinnin suorittamiselle ilmenivät resurssit, joita työn tekemiseen tarvitaan. Automatisoitu testaus on investointi, joka tarvitsee tukensa ylemmältä johdolta. Sen säästöt saadaan pidemmällä aikavälillä.

Kasurinen (2013) toteaa, että testaus on kannattavuuden kannalta tärkein työvaihe, sillä kannattavuuden ja testauksen välillä on selkeä yhteys. Yritys, joka testaa tuotteensa huolellisesti saa paremman katteen, ja kun tähän lisätään vielä mahdollinen asiakaskato ja mainevahingot, ero kasvaa entisestään. Virheen korjaaminen tekovaiheessa maksaa 1-2 prosenttia siitä mitä se maksaa, kun julkaisu on jo tehty.

Testaus miellettiin yrityksessä hyvin perinteisellä tavalla, yhtenä työvaiheena vesiputousmäärittelyssä. Tätä ajattelutapaa pyrittiin projektissa muuttamaan, sillä vesiputousmallissa testaus alkaa niin myöhään, että virheitä on jo tapahtunut paljon, ja virheiden korjaamiseksi voidaan joutua tekemään paljon työtä. Tämän vuoksi testaustoimenpiteitä pyritään vastuuttamaan määrittelystä alkaen aina ylläpitoon asti.

Kuviossa 3 kuvataan perinteinen vesiputousmalli, jonka mukaista toiminta oli ennen automaatiotestaustoimien mukaan ottamista kehitystyöhön. Määrittelyvaiheen laajuus lähtötilanteessa vaihteli paljon työstä riippuen, mutta se saattoi olla informaatioltaan puutteellinen, ja suunnittelu ja toteutus vaihe limittyivät toisiinsa. Testaus tuli mukaan vasta kun ominaisuus oli jo valmis.

KUVIO 3. Perinteinen vesiputousmalli (Muokattu Kasurinen 2013)

Muutosta lähdettiin aluksi viemään v-mallilla, joka jatkaa vesiputousmallia. Testaus alkaa toteutusvaiheessa ja toimintamallin sijasta laajennetaan ensin testauksen kattavuutta ja testauskohteita. Käytännössä tämä tarkoittaa, että kehitysvaiheessa kehittäjä tekee yksikkötestit, työstä laaditaan integraatiotestit, jonka jälkeen manuaalisesti suoritetaan sisäinen järjestelmätestaus, ja ennen julkaisua hyväksymistestaus. Tavoitteena ei kuitenkaan ollut jäädä v-malliin, sillä sen keskeinen ongelma on, kuten monessa muussakin ohjelmistotuotannon mallissa, että testaus alkaa liian myöhään. Siksi alusta alkaen, vaikka askeleet tehtiin pieniksi, testaus ja testausnäkökulma tuodaan mukaan aina vain varhaisemmassa vaiheessa, ja viestinnällä ja koulutuksella välitetään ajatusmallia, että testit voidaan kirjoittaa jo ennen toteutusta, eli painotetaan testauslähtöisestä kehittämisestä (TDD, Test Driven Development) Tämän tason jälkeen on helpompi ottaa mukaan käyttäytymislähtöinen kehittämiseen, jossa käyttötapaukset määritellään ensimmäisenä, ja käyttötapaus kirjoitetaan ennen kuin ominaisuus kehitetään.

Toimintamallin jalkauttamisessa otetaan ideologista mallia RUP-mallista, jonka ero vesiputoukseen on, että vaiheittaista kehitystapaa hyödynnetään tehokkaammin ja testaus tulee mukaan alusta loppuun. Keskeinen ajatus on se, että sen sijaan että määrittely, suunnittelu, toteutus ja testaus tehtäisiin vuoron perään, niitä kaikkia tehdään koko ajan. Päätyövaiheiksi muodostuvat aloittaminen, tarkentaminen, rakentaminen ja siirtymä. Aloittamisvaiheessa tehdään työmääräarviot ja päätetään kannattaako ominaisuus tai projekti

ylipäätään toteuttaa. Tarkentamisvaiheessa arkkitehtuurin ja kuvauksen merkitys on suurin, jotta rakentamisvaiheessa toteutus etenee ilman kysymyksiä ja siirtymävaiheessa toteutus voidaan ottaa hallitusti käyttöön. (Kasurinen 2013, 15-16.)

Muutokset toimintaan alkavat jo määrittelyvaiheesta, sillä toiminnallisuuksia tarkastellaan käyttäjätarinoiden, eli loppukäyttäjän toimintaskenaarioiden näkökulmasta. McPeak (2018) korostaa kuinka kommunikaatio paranee käyttäytymisläheisessä kehittämisessä (BDD, Behaviour Driven Development). Määrittely syntyy kolmen roolin yhteistyön tuloksena: asiakas tai tuoteomistaja kertoo vision käyttäjätarinan kautta, jonka kehittäjä määrittelee teknisiksi tarpeiksi, jolloin testaaja varmentaa, että uusi ominaisuus ratkaisee tarpeen, miksi ominaisuus tehdään.

Käyttäjätarinat ja käyttäytymiskuvaukset määritellään selkokielellä, jolloin minkä tahansa roolin henkilö voi helposti varmistaa, miten toiminnon halutaan toimivan ja millaiset käyttäytymistilanteet suunnittelussa on huomioitu. Näkökulma muuttuu, kun ei vain mietitä toimiiko tietty toiminto, vaan pohditaan, miten tiettyä toimintoa käytetään kyseisessä skenaariossa. Skenaariot muodostavat pohjan hyväksymistesteille, joiden automatisointi kasvattaa nopeasti testikattavuutta. Testiautomaatio liitetään osaksi jatkuvaa integraatiota. Tyypillinen virhe käyttöönotossa on, että käyttötarinoiden suunnitteluun ei varata aikaa (Gharhai 2015).

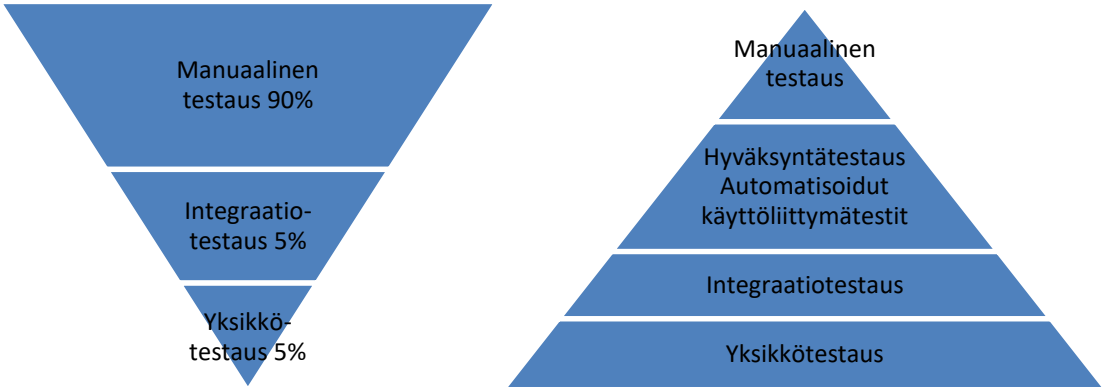
Testausvaihe muuttaa luonnettaan siten, että manuaalisen testauksen rinnalle tulee automatisoitu integraatiotestaus, jotka lisätään jatkuvaan integraatioon. Jatkuva integraatio muuttaa toimintamallia siten, että työtä testataan regression havaitsemiseksi myös silloin kun suoraan kyseiseen toimintoon ei tehdä muutoksia.

Automaatiotestauksesta keskustelussa ilmeni muutamia selkeitä haasteita. Päällimmäisenä oli huoli siitä miten paljon testien tekeminen kuormittaa jo nyt kiireisissä projekteissa. Kun ymmärrys automaatiotestauksen tarpeesta parani, kasvoivat odotukset sen suhteen mitä automatisoinnilla voidaan saavuttaa.

Automaatiotestaus ei takaa tuotteen virheettömyyttä, mutta sen tavoitteena on varmistaa, että virheet ovat harvinaisia ja luonteeltaan poikkeuksellisia, niin että ne eivät haittaa tuotteen normaalia käyttöä. Toisaalta testaus itsessään ei takaa laatua, pikemminkin tuo

ihmiset tietoiseksi siitä. Keskeisin laadun tekijä on suunnittelu- ja kehitysvaihe, ja siksi testaus ja arviointi haluttiin jo osaksi näitä vaiheita.

Kuviossa 4 on esitetty testauksen nykytila ja tavoitetila. Nykytilassa korostuu manuaalisen testauksen suuri osuus ja näkyvät integraatio- ja automatisointiosuudet ovat suppealta alalta ja keskittyvät muutamien ydintoiminnallisuuden suorituskyvyn testaukseen sekä manuaalisen testauksen tukiprosesseihin, joilla osa toiminnasta tehdään automatisoidusti sovittujen tietojen mukaan, mutta jossa tarkastukset silti tehdään aina manuaalisesti odotetun tuloslistan kanssa. Tavoitetilassa pohjana on helposti ylläpidettävät ja kustannustehokkaat yksikkötestit. Integraatiotestit varmistavat, että eri sovellusten osat ja tasot toimivat yhteen toivotulla tavalla. Automatisoidut hyväksyntätestit toimivat käyttöliittymätasolla käyttöönottoprojektissa, ja laajenevat rajapintoihin. Manuaalisen testauksen osuus on jäävuoren tärkeä huippu, tutkivaa testausta, uusien ominaisuuksien testausta, toiminnan muutoksen testausta ja käytettävyyteen.



KUVIO 4. Testauksen nykytila ja tavoitetila

Malli toimi erinomaisesti viestinnän apuvälineenä, koska sillä oli helppo osoittaa testaajille, että kyse ei ole heidän tärkeän työnsä korvaamisesta, vaan työtaakan tasoittamisesta ja runsaasti toistoa vaativien toimien vähentämisestä. Mallin avulla oli helppo selittää päättäjille miksi tämä tuo meille riskienhallinnollista turvaa, kustannussäästöjä, joustavuutta ja nopeutta. Mallia käytettiin havainnollistaan kehittäjille, miten paljon luottavaisemmin koodia voidaan muuttaa ja kehittää; voidaan ottaa uusia teknologioita ja versioita käyttöön lyhyemmässä ajassa, kun voidaan nopeasti varmistaa sovelluksen yleinen toimivuus muutoksen jälkeen.

Tahtotila työn eteenpäin viemiseen saattaa vaihdella projektin aikana, ja tällöin on tärkeää pystyä nopeasti muistuttamaan syyt, miksi muutosta ollaan tekemässä, ja miksi se kannattaa.

2.6 Testausorganisaatio ja vastuun jako

Ketterä kehittäminen, scrum, DevOps ja jatkuva integraatio muuttavat tapoja, joilla tuotekehitystä tehdään. Kun tavoitteena on nopeaan julkaisuun pystyvä tuotekehitystiimi, pitää testausstrategiat ja vastuut käydä läpi. Testausta tulisi tehdä eri näkökulmista. Esimerkiksi tuotekehitystiimi tuntee eri teknologiat ja alustat, joten heidän tulisi kiinnittää huomiota teknologian testaukseen ja välittää tietoa kuinka uusia teknologioita kannattaa testata.

Modernissa organisaatiossa testaus on tehtävä, jota voi tehdä kuka vain, eikä se ole enää vain testaajien tehtävä. Jaettu vastuu tuotekehityksen laadusta vaikuttaa tiimin jäsenten osaamisprofiiliin. Testaajille tämä tarkoittaa tarvetta ymmärtää teknisiä riskejä ja vaatimuksia. Kehittäjille testaaminen ja laatukeskeinen toiminta tarkoittaa kriittisyyttä oman koodin lisäksi itse määrittelyyn. (Simbhoedatpanday 2015.)

Automaatiotestauksen käyttöönotto pitää huomioida optimaalisen testausresurssimäärän arvioinnissa. Kohdeorganisaatiossa testaustyössä on aina mukana ainakin kolme tahoa: testauspäällikkö, testaajat ja kehittäjät. Testauspäällikkö voi olla testauksesta vastaava esimies, joka on koko projektin tai tuoteosion päällikkö. Testaajat voivat olla joko kokopäiväisiä tai tehdä muitakin työtehtäviä kuten ohjelmointia tai käyttöliittymäsuunnittelua. (Kasurinen 2013, 58.)

Automaatiotestauksen osalta resurssointi tuli kuitenkin miettiä erikseen, sillä manuaalisen testauksen tekijöillä ei ollut aikaa automaatiotestaukseen ja toisaalta automaatiotestaus on luonteeltaan ja tarpeeltaan erilainen. Automaatiotestausroolin tehtäviin kuuluu tiimin ohjaaminen kohti jatkuvan julkaisun mahdollistamista. Käytännössä laadun valvonnan kannalta rooli on samankaltainen testaajan roolin kanssa, mutta automaatiotestaajan on osattava ohjelmoida jonkin verran ja ymmärrettävä sekä kehitystiimin tarpeita että toimialan ja asiakkaan vaatimuksia. Automaatiotestaaja monitoroi koodimuutoksia ja jatkuvasti ajossa olevia testejä. Automaatiotestaustiimin jäsenten tulee pysyä ajan tasalla

uusista teknologioista ja niiden teknisistä riskeistä. Automaatiotestauksen avulla testaustiimi pystyy poistamaan pullonkauloja prosessista ja nopeuttaa palautteen saamista. Testaajat laadunvalvojina ohjaavat tiimiä ratkaisemaan perimmäiset syyt sen sijaan että korjattaisiin vain oireita. (Simbhoedatpanday 2015.)

Automaatiotestaus suunniteltiin otettavaksi eri tuotteisiin eri aikaisesti käyttöön. Tavoite on, että työ alkaa kickoff-palaverissa, jossa luodaan pohja testaussuunnitelman laatimiseen. Ensimmäisellä kertaa otetaan kantaa laajemmin vastuisiin ja testaustapoihin, joihin myöhemmin viitattiin vain, kun toimintaa muutettiin, tai oli tarve käydä läpi mitä oli sovittu. Testauspäällikkö, kehittäjät ja testaajat luovat yhdessä testaussuunnitelman ja laativat ensimmäiset testitapaukset, jotta työ päästään aloittamaan ja tiedetään, kuka tekee mitä ja missä vaiheessa.

Lähtökohtaisesti rooleja mietittiin testivaiheiden kautta. Yksikkötestit integraatiotestien implementointi ovat lähtökohtaisesti kehittäjien tehtäviä. Testaajat ja kehittäjät suunnittelevat yhdessä testitapauksia. Testaajat suorittavat määritellyt testitapaukset, raportoivat ja virheen tapahtuessa ilmoittavat virheestä ja arvioivat ongelman vakavuuden.

Oikeiden henkilöiden valinta oikeisiin asioihin on kuitenkin kriittinen osa projektin onnistumista. Jokaisessa scrum-tiimissä tulisi olla ainakin yksi edelläkävijä, joka perehtyy automaatiotestaukseen hieman nopeammalla syklillä ja syvällisemmin kuin muut, jotta pystyy tukemaan muuta tiimiä testauksen automatisoinnissa. Tätä varten käyttöönoton kokemusten perusteella päädyimme siihen, että tästä on hyvä tehdä oma roolinsa. Championin ominaisuuksia ovat hyvä tuntemus tiimin kehittämästä tuotteesta ja tietokannasta ja laadunhallintatoimista. Mutta avainasemassa on halu ja kiinnostus viedä automatisointia eteenpäin.

Tiimin edelläkävijät (championit) ja automaatiotestauksen asiantuntijat muodostavat yhdessä tiimin, joka vie käyttöönottoa eteenpäin ja kehittää toimintamalleja, jolloin tuotteen osaajien taidot yhdistetään automaatiotestauksen asiantuntijan osaamiseen. Jos automaatiotestauksen asiantuntijuus ei ole osa tiimin jäsenten osaamisportfoliota, tämä on oikea rooli myös kolmannen osapuolen osallistumiselle, jolloin käyttöönotossa automaatiotestauksen asiantuntijuus tulee ulkoiselta konsultilta.

Edelläkävijän ei ole välttämätöntä olla testaaja. Tuotteen kehittäjät ovat ketterässä kehittämisessä mukana miettimässä, mitä asiakas haluaa ja tarvitsee, aikataulutusta ja tapoja tehdä ne. Kun automaattitestauksen näkökulma on vaatimuksena olemassa, se ohjaa ohjelmoimaan helposti testattavaa koodia, modulaarisesti ja yhtenäisillä ID- ja UI-rakenteilla.

Testien luontivaihe on ensimmäinen vaihe, mutta vastuutuksen pitää yltää myös testien ylläpitoon ja analysointiin sekä itse testien ajoon. Tiimin on ratkaistava, kenellä on keskitetysti kuva testeistä ja sitä kautta kattavuudesta. Kuka raportoi testituloksista? Kuka on käyttöön valittujen työkalujen osaaja, joka osaa antaa talon sisällä tukea työkalun käytössä? Kun tiimi koostuu eri roolien osaajista, sitoutuminen laatuun ja sen varmistamiseen koskee kaikkia tiimin jäseniä, niin kehittäjiä kuin testaajia.

3 AUTOMAATIOTESTAUKSEN KÄYTTÖÖNOTTO

3.1 Automaatiotestauksen käyttöönoton suunnittelu

Testausautomaatio ei korjaa rikkinäistä prosessia, joten projektin aikana keskusteltiin paljon paitsi automaatiotestauksen suhteesta prosessiin, myös prosessista itsestään. Toisaalta tunnistettiin tuotekehitystiimeissä tarvetta perehdytykseen ja koulutukseen, ja pohdittiin, onko ratkaisuna erillinen automaatiotestaustiimi vai tiimien kouluttaminen automaatiotestaukseen.

Nykytila-arvioinnin perusteella päätettiin, että ensimmäisessä vaiheessa automaatiotestaus on syytä käsitellä omana projektinaan, jonka osana on tuotekehitystiimien perehdyttäminen uuteen toimintamalliin. Tavoitteena on sovittaa ja hallinnoida testejä kehitystiimin sykleissä.

Automatisoidun testauksen suunnittelu lähtee strategian luonnista. Strategiassa otetaan kantaa siihen, mitkä vaiheet ja testitapaukset halutaan automatisoida. Koska ja kenen toimesta automatisointipäätös tehdään? Miten testiautomaatio on organisoitu, kuka toteuttaa, käyttää, tukee ja ylläpitää? Miten investointi rahoitetaan? Miten automaatiotestaus kytkeytyy muihin toimintoihin, mitä vaatimuksia ne asettavat automatisoinnille?

Seuraava vaihe on päättää testiympäristön tuki testiautomaatiolle. Onko testit pysyvästi tarjolla, onko ympäristö jaettu muiden toimien kanssa, onko automaatiotestaukselle nimetty oma ympäristö? Vai luodaanko testiympäristö aina tarpeen mukaan?

Korkealla tasolla automaatiotestien luomiselle, hallinnoinnille ja ajolle määriteltiin tavoitteiksi helppokäyttöisyys, ylläpidettävyys ja raportoitavuus. Nämä vaatimukset ulottuvat testausautomaatiotyökalun valintaan laajentuen käytössä olevien teknologioiden tukeen. Testauksen käyttöönotolle muodostui suunnittelun aikana kolme selkeää kokonaisuutta 1) työnkulun eli prosessien läpikäynti, 2) koulutus ja tiimin oppimisen tukeminen ja 3) teknisten valmiuksien ratkaisu automaatiota varten.

Nykytilan arvioinnissa nousi esiin vaatimus, että testien hallintaa tulee tehdä ominaisuusvetoisesti. Tämä tarkoittaa, että jos useampi tuote käyttää samaa ominaisuutta, ei ole tarkoituksenmukaista toistaa tavoitteeltaan samoja testejä useaan paikkaan. Eli esimerkiksi ominaisuus, joka on toteutettu palvelimelle, testataan ominaisuuden kannalta palvelinrajapinnassa. Tuotteiden kautta ajateltuna kyse on integraatio- ja järjestelmätasosta, jossa varmistetaan, että toiminallisuus toimii myös toisen sovelluksen kautta. Eli testauksen tavoite, syy miksi testi tehdään, on eri. Siksi testauksenhallinnan kannalta on parasta, jos testauksen syy ja kohde dokumentoidaan testille.

Testitapausten suunnittelu määrittää, miten jonkin ohjelmiston osa testataan. Testitapausten suunnittelun pitäisi tuottaa testejä, joilla on tietyt arvot testin tarvitsemalle oletustilalle, lähetettävällä tiedolle ja oletetuille tuloksille. Jokaiselle ohjelmistolle on olemassa suuri määrä mahdollisia testejä ja näistä testeistä tulisi valita automatisoitaviksi vain sellaiset, joilla saadaan havaittua mahdollisimman suuri virhemäärä testattavasta ohjelmistosta. Testejä ei kannata automatisoida tilanteissa, joissa testejä ajetaan todella harvoin tai tuloksien tarkistaminen ihmisiltä onnistuu helposti, mutta ohjelmallisesti se on todella vaikeata. (Fewster & Graham 1999, 14.)

Yhteisiä pelisääntöjä määriteltäessä havaittiin tarve määritellä yksinkertaisia tiivistyksiä ja ohjaavia rakenteita. Esimerkiksi määrittelyn tasoon minimivaatimus on, että siihen löytyy esimerkkiskenaario ja tiedot siitä, minkä ongelman ominaisuus ratkaisee, eli miksi ominaisuus tehdään. Ongelmat vaatimuksissa voivat ilmetä monin eri tavoin, mutta ydinongelmaksi lähtötilanteessa arvioitiin, että tiketistä puuttui paljon oleellista tietoa. Käytännössä tuoteomistaja tai muu tiketin omistaja ei osannut tai ehtinyt muuttaa tietämystä tasolle, jolla tiimi voisi todella työskennellä. Kun tiedot eivät ole keskitetysti yhdessä paikassa, niistä viestiminen vaikeutuu merkittävästi.

Ominaisuuden määrittelylle asetettiin tavoitteeksi käyttötapaustarkkuus, eli kuvaus siitä, mitä järjestelmän pitäisi tehdä, kun sitä käytetään määritellyllä tavalla. Suunnittelussa on hyvä tehdä riskikartoitus komponenteista ja sen avulla keskittää testausta kriittisimpiin pisteisiin. Määrittelyn pitäisi olla myös sellaisella tasolla, että ominaisuuden koko ominaisuudet ovat luettavissa tiketiltä. Jotta tämä määrittely toteutuisi, päätettiin pilotoida ratkaisua, jossa integraatiotestien käyttötapaukset laaditaan jo tiketin luontivaiheessa, jolloin ne ovat kehittäjän hyödynnettävissä ja implementoitavissa jo kehittämisvaiheessa. Kun tuotekehityksen malli on ketterä ja näkökulma käyttäytymisläheinen, määrittely ei

pääse toteutukseen ennen kuin tiimillä on ymmärrys toteutettavasta ominaisuudesta. Kun päämäärä on jaettu, voidaan määritellä tarkentavat skenaariot ja osittaa työ skenaarioiden ja niistä muotoutuvien vaatimusten kautta. (Kasurinen 2013, 63.)

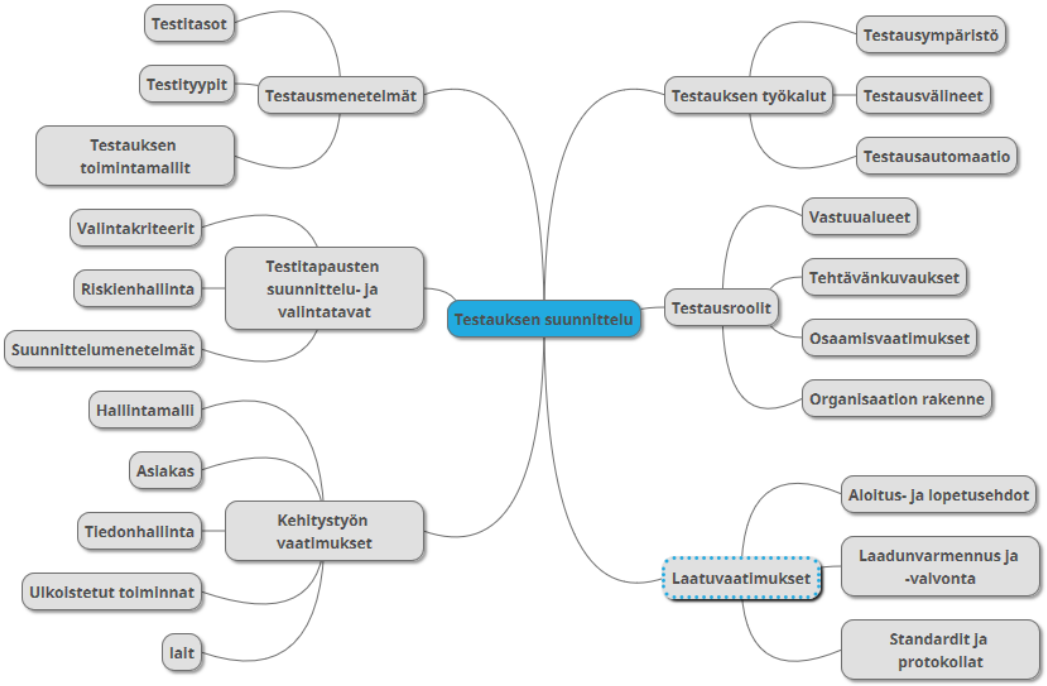
Testauksen suunnittelussa hahmoteltiin ensin testauksen tasot, testauksen mittarit, prosessi ja työkalut, joiden jälkeen jokaista aluetta laajennettiin. Käytettävien termien määrittelyn jälkeen muodostui sanasto, jonka perusteella testauksen avainsanoja on helppo lisätä. Testien dokumentointitapaan luotiin säännöt, joilla testien rajoitteet merkitään. Suunnittelu tehtiin puhtaasti automaatiotestaukseen, sillä manuaaliseen testaukseen on jo olemassa vakiintuneet ja toimivat toimintatavat, joita ei haluttu käyttöönottovaiheessa muuttaa, vaan muutoksia käyttöönotetaan vaiheittain.

Testausta lähdetään automatisoimaan jo olemassa olevaan tuoteperheeseen, joten suunnittelun keskeisiä kohteita on varmistaa testaus oikeaan aikaan ja mahdollisimman aikaisin. Tuotteen ollessa laaja, koodikattavuuden saavuttaminen vie paljon resursseja. Testausautomaatiota otetaan mukaan kolmella strategialla: uuskehityksessä automaatio otetaan testaukseen heti mukaan, refaktorointiin meneviin ominaisuuksiin tehdään automaatiotestit ennen muutoksen toteuttamista ja lisäksi laaditaan perustoiminnallisuuksista lista, jonka kautta kattavuutta lisätään jälkikäteen automatisoinnilla testauskohteen kriittisyyteen perustuvalla prioriteettijärjestyksellä. Testauksen painopiste kohdistetaan seuraavien periaatteiden mukaan: 1) Kohteen kriittisyys, eli mikäli virhe kohteessa on asiakkaan kannalta kriittinen, nostaa se heti testauksen prioriteetin korkealle. 2) Vikojen kasaantuminen, eli painopistettä asetetaan sinne missä tiedetään olevan eniten vikoja.

Kuviossa 5 kuvataan testauksen suunnittelussa käytetty runko ajatuskarttana. Ajatuskarttaa käytettiin projektissa kommunikaatiovälineenä, jonka perusteella testauksen suunnitteluun otettiin automaatiotestausnäkökulma mukaan. Mallia uudelleenkäytetään, kun testauksen piiriin otetaan uusi tuote tai isompi kokonaisuus. Kehitystyön vaatimukset tulevat asiakkaalta tai viranomaisilta lakien ja ohjeistuksien muodossa. Ulkoistetut toiminnot ja niiden rajaehdot muodostavat vaatimuksia testien suunnittelulle. Kehitystyön hallintamalli ja tiedonhallinta otetaan huomioon testaussuunnittelussa.

Käytännössä käydään läpi kohteen testausmenetelmät, ja valitaan testauksen taso, ja siihen sopivat toimintamallit suhteessa testaustyyppeihin. Näiden perusteella voidaan valita

työkalut, jotka vaikuttavat testausympäristön ja välineiden valintaan, sekä päätökseen automatisoinnin piiriin tuomisesta. Testitapausten suunnittelu- ja valintatapojen määrittelyssä otetaan kantaa valintakriteereihin ja riskienhallintaan, sekä siihen millaisia suunnittelumenetelmiä käytetään. Näiden perusteella voidaan tunnistaa testausroolit, niiden tehtävät ja vastuualueet sekä osaamistarpeen, jolloin voidaan ottaa kantaa organisaatiorakenteeseen. Laatuvaatimuksessa otetaan kantaa standardeihin ja protokolliin, ja laadunvarmennustapoihin ja niiden noudattamisen valvontaan, jolloin määritellään aloitus- ja lopetusehdot.



KUVIO 5 Automaatiotestauksen suunnittelun vaatimukset projektissa

Testaustasolla viitataan tässä dokumentissa kategorioihin, joilla testauksen kohteita, eli testattavia osakokonaisuuksia, käsitellään. Testaustasossa otetaan kantaa siihen, kohdistuuko testaus yksittäiseen moduuliin, moduulien yhdistämiseen vai koko järjestelmän kattavaan testaukseen. Testaustasoon liittyy kiinteästi testaustapa, eli onko kyse turvallisuustestauksesta, käytettävyystestauksesta tai vaikkapa rasitustestauksesta.

Tyypilliset testauksen vaiheet ovat 1) kehittäjän tekemä yksikkö- ja integraatiotestaus, 2) toiminnallinen testaus, 3) järjestelmätason testaus, 4) suorituskyvyn testaus, 5) testaajan integrointitestaus, 6) hyväksyntätestaus ja 7) tietoturvatestaus. Tässä dokumentissa keskitytään yksikkö- ja hyväksyntätestaukseen sekä sivutaan järjestelmätestausta, koska

käyttöönotossa ensimmäisiksi automatisointikohteiksi valittiin yksikkö- ja hyväksyntätestaus. Tietoturvatestaus rajattiin ulkopuolelle käyttöönottovaiheessa.

Käyttöönoton suunnittelu tiivistyy automaation käyttöönoton projektisuunnitelmaan. Koska tuotteita oli useita, päätettiin viestinnän avuksi koostaa yleiskäytettävä kysymyslista, jonka tavoitteena on varmistaa, että projekti on realistinen ja mitattava. Käynnistysprojektin suunnittelupalaverissa otetaan kantaa siihen,

- mitä automatisoitu testaus tarkoittaa organisaatiossa?
- mitä ongelmia yritetään ratkaista automatisoinnilla?
- mitkä ovat organisaation tavoitteet automatisointiprojektissa?
- miten testien automatisointitavoitteet tukevat testauksen tavoitteita kokonaisuudessa?
- miten manuaalinen testaus sopii automaatiotestaussuunnitelmaan?
- mitkä ovat testiautomaation koodikattavuusodotukset?

Mitattavalle tavoitteelle on hyvä asettaa lisämääre, joka määrittää todellisen arvon, jota tavoitellaan. Esimerkiksi projektin ensimmäisen vaiheen tavoite on, että hyväksyntätestauksen kesto vähenee 30% kun osa testeistä korvataan automatisoiduilla. Regressiotestauksen keston lyhenemisen takia voimme onnistumiseen lisätä arvon nousuksi sen, että testaajien aikaa vapautuu tutkivaan testaukseen tai että henkilöresurssien käyttö kevenee tilanteissa, joissa sovellus on vakaa ja suoriutuu savu- ja hyväksyntätesteistä.

Projektin mittareiden määrittelyssä on syytä ottaa kantaa jo käytössä oleviin mittareihin, kuten raportoitujen havaintojen määrään, niiden ratkaisuaikoihin, testien suorituksen kestoon, koodikattavuuteen ja resurssien käyttöön. Olemassa olevien mittarien käytön etu on, että dataa on käytettävissä ennen testausautomaatiota.

Projektisuunnitelman ja käyttöönottoprojektin päätuotoksena on työkalujen valinta, työkalujen käyttöönoton toteutumisen tulokset, automaattitestien rakenne tai soveltuvuus testin, eli proof of concept -testin yhteydessä muodostuneet testitapaukset. Käyttöönottoprojektissa huomio kiinnittyi eniten aluksi kriittisimmän testauskohteen testien etenemiseen.

Resurssien määrittelyn lisäksi on suunniteltava, miten toimitaan, jos projektia uhkaavia riskejä toteutuu, esimerkiksi jos projektin toteuttajien resursseissa tapahtuu muutoksia.

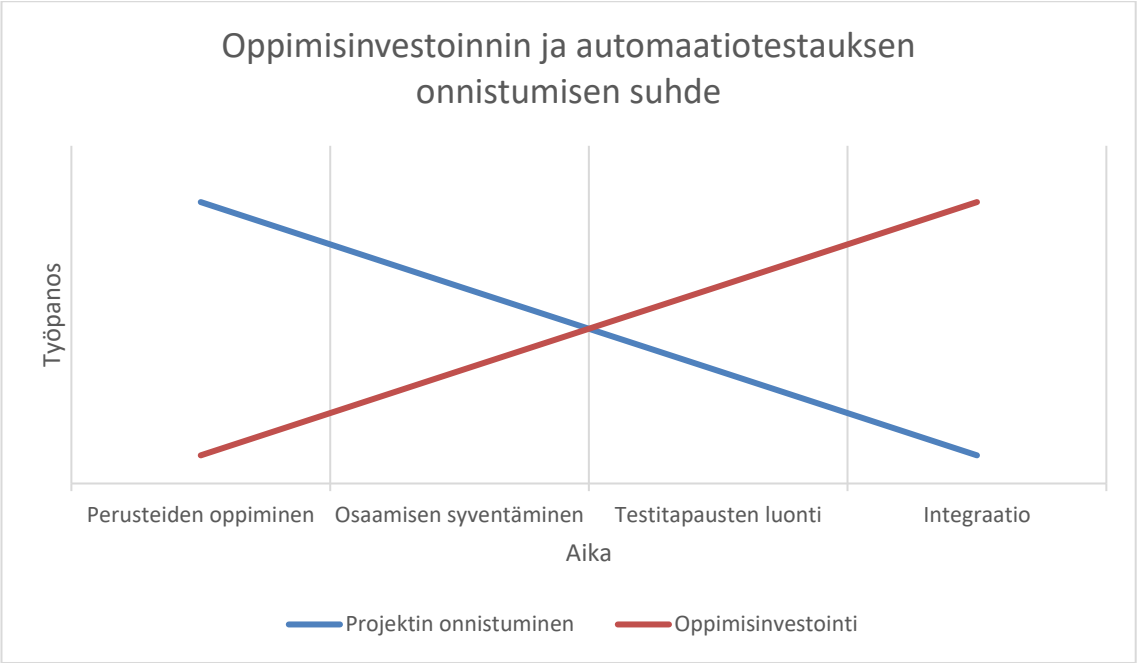
Osa käyttöönoton suunnittelua on nykytilan arviointi, jossa huomio kohdistuu tiimien osaamistasoon, testauksen nykytilaan ja tuotekehitysprosessiin.

Käyttöönottoprojekti jaettiin vaiheisiin, joihin sisältyy testauksen lisäksi nykyprosessien analysointi, toimintasuunnitelman hiominen, tiimijäsenien tehtävien määrittely ja kulttuurimuutos organisaatiossa. Käyttöönotossa valmisteltiin neljä vaihetta, joista ensimmäinen keskittyy analysointiin ja jatkuvan integraation käyttöönoton suunnitteluun, toinen testitapausten valmisteluun ja jatkuvan integraation tuomiseen ja kolmas automaatiotestien tekoon, suoritukseen ja prosessisuunnitteluun. Neljännen vaiheen tavoite on tiedon dokumentointi, testirakenteiden hienosäätö ja tiedon jakaminen. Toteutus edellyttää, että tekniset ratkaisut ja ympäristöt on ratkaistu ennen sprinttien alkua tai viimeistään ensimmäisessä sprintissä.

Testitapausten keskitetty sijainti on hyvä määritellä jo projektisuunnitelmaan, mikäli sijainti on tiedossa. Käyttöönottoprojektissa paikka tarkentui projektin edetessä. Kuten projektille yleensä myös automaatiolle on hyvä koota oma tiimi, mielellään mahdollisimman lähelle muuta toimintaa, jolloin kommunikointi on helppoa. Projektisuunnitelmaan olisi voitu sisällyttää myös missä ja milloin koulutus tehdään, mutta käyttöönoton suunnittelussa tämä tieto jäi avoimeksi ja suunniteltiin määrittelyn edetessä, jolloin tietämys tarvittavan koulutuksen määrästä tarkentui.

Koska sekä työkalu että toimintatapa ovat tiimille uusia, on tärkeää aikatauluttaa aikaa oppimiseen paljon. Koulutus on syytä ajoittaa hetkeen, jolla testausta voidaan saman tien aloittaa hyödyntämään, koska käydyt asiat ovat tuoreina mielessä ja motivaatio muutosten tekoon on korkealla. Harjoitukselle on yleensä tarvetta, vaikka automaatiotestaus olisi tuttua jollain toisella työkalulla, sillä ihmisillä on tapana siirtää opitut toimintatavat. Jokainen automatisointityökalu on kuitenkin uniikki, joten parhaat toimintatavat voivat olla erilaiset eri tuotteilla.

Kuviossa 6 on kuvattu resurssien tarve suhteessa aikaan eri vaiheissa suhteessa aikaan ja sen vaikutus toimintatavan jalkautumiseen. Alkuvaiheessa, kun perusteita opetellaan, automaatiotestaus vie paljon aikaa. Investointi on suuri. Kun osaaminen kehittyy, testien luominen rutinoituu ja aika yksittäiseen testiin pienenee.



KUVIO 6. Käyttöönottoprosessin oppimiskäyrä vaiheittain (Muokattu, Kasurinen 2013)

3.2 Muutoksen hallinta testauksen käyttöönotossa

Halu testauksen automatisointiin syntyi alun perin esimiesten tarkastellessa toimintaympäristöä ja tilanteen kehitystä. Tilanteen muuttuminen ennakoitiin haasteellisemmaksi ilman toimenpiteitä muutoksen läpivientiä ryhdyttiin suunnitelemaan ja ajamaan. Vaikeaksi tilanteen teki, että vaikka tarve tunnistettiin, toimenpiteiden laajuutta ja merkitystä käytännön työhön ei oltu avattu kunnolla. Tilanteen aiheuttama epävarmuus ja toiminnan lamaantuminen otettiin hallintaan tuen ja tiedotuksen kautta.

Ihmiset kokevat muutokset eri tavoin, ja siksi joillekin pieni toimintatapamuutos voi olla isompi asia kuin vaikka organisaation rakenteita muuttava muutos (Ponteva 2010, 9). Tarpeen tunnistaminen on ensimmäinen askel, mutta johdon tulee selvästi osoittaa sitoutumisensa paitsi osoittamalla, että muutosta tarvitaan, myös ymmärtämällä olosuhteita, joissa muutos tapahtuu. Johdon tehtävä on toimia esteiden raivaajana ja kannustajana tarjoten muutokselle tarvittavat resurssit. (Ponteva 2010, 10-11).

Automaatiotestauksen aloitukselle on tärkeä varata alusta alkaen hyvin resursseja, ja varmistaa että projektin tekijät pystyvät keskittymään automaatiotestaukseen täysin. Tilanteissa joissa samasta resurssista kilpaili monet, kiireellistä huomiota vaativat työt, automaatiotestaus jää helposti taka-alalle. Jos työskentelyrauhaa ja resurssien toteutumista ei

ole varmistettu, työ pysähtyy helposti. Onnistuneen muutoksen kannalta on elintärkeää kehittää tiimin oppimiskykyä, ja jos tiimin jäseniä ei saa kuormittaa testauksen käyttöönottoon liittyvillä asioilla, suunnittelu jää vain puolittaiseksi.

Jatkuvassa muutoksessa tehtyjä ja päätettyjä asioita tulee jatkuvasti tarkastella ja tarvittaessa tehdä korjausliikkeitä. Tämä on henkilöstölle erityisen raskasta, jos he eivät tiedä muutoksista ennen päätöstä, sillä silloin muutos tuntuu suunnittelemattomalta ja poukkoilevalta ja esimiehen täytyy paitsi kuunnella henkilöstöä, myös kirkastaa muutostarpeiden syitä ja saada heidät ymmärtämään motiivit muutosten takana. (Ponteva 2010, 12.)

Osa testauksen käyttöönottoa on uusi tapa organisoida ja työskennellä. Johdon sitoutumisen ollessa kunnossa, itse työntekijöiden tulisi innostua muutoksesta, mutta ketään ei voi pakottaa muuttumaan. Olennaista on, että kaikki viestivät samaa asiaa: muutos on myönteinen ja toivottu asia (Ponteva 2010, 18).

Motivaation nostatuksessa organisaation esimiehet onnistuivat hyvin projektissa mukana olleiden henkilöiden osalta, mikä edesauttoi merkittävästi testausasenteiden muuttumista positiiviseen suuntaan valmistellen otollista pohjaa käyttöönotolle.

Ponteva (2010) toteaa organisaatioiden tyypilliseksi haasteeksi sen, että koskaan ei tunnu olevan tarpeeksi aikaa pysähtyä ja tehdä asiat kunnolla. Ei ehditä pysähtyä miettimään tärkeysjärjestystä. Ongelma korostuu, jos esimiehellä on niin kiire, että alaisille ei ole aikaa vaan erilaiset - sinänsä tärkeät ja välttämättömät työt kuten kokoukset, sähköpostit ja soittopyynnöt - vievät kaiken ajan. Tällöin esimiehen lyhyet ja vapaamuotoisetkin tapaamiset voivat olla avainroolissa organisaation kehittymiselle. Esimiehen tilannetta vaikeuttaa, jos liian moni rauta on samaan aikaan tulessa. Tällöin hyvä ratkaisu on ottaa aikalisä, ja jaksottaa muutoksia mahdollisuuksien mukaan, jotta tavoiteltavat muutokset ovat realistisia. Kaikkeen ei pysty, vaikka kuinka tahtoisi.

Ihmisten kanssa jutteleminen on tärkeää muutoksen keskellä. Ilman ymmärrystä ja käsitystä muuttamisen mahdollisuuksista, koko muutos vaarantuu. Yleisen tiedotuksen lisäksi keskustelua on hyvä käydä myös pienemmissä ryhmissä, sillä myös ryhmillä on tunteet, joihin useiden yksilöiden tunteet vaikuttavat. Työyhteisön tunteiden käsittelyn tekee haasteelliseksi se, että toiminnassa on tietoinen ja tiedostomaton taso. (Ponteva 2010, 29-30.)

Muutoksen kokemiseen vaikuttaa kuinka sitoutunut on työhönsä ja samaistunut organisaatioon. Samaistumisella tarkoitetaan sitä, haluaako yksilö olla osa organisaatiota. Kun henkilö pitää työstään ja työpaikastaan, hän on innokas kehittämään sitä. Jos henkilö haluaa olla organisaatiossa, mutta on etääntynyt työtehtävistään, hän kyllä osallistuu muutokseen, mutta ei itse edesauta muutosta. Vieraantuminen työstä ja organisaatiosta näkyy välinpitämättömyytenä ja kyynisyytenä. Toisaalta omapäisyys ja kapinallisuus kielivät tilasta, jossa oma työ antaa aihetta ylpeyteen, mutta organisaatio sinänsä ei merkitse yksilölle paljoa. (Ponteva 2010, 36-37.)

Yksilön innostuneisuus ja kiinnittyminen organisaatioon ei tarkoita, että kaikki muutokset olisivat mieluisia. Lisäksi usko muutokseen voi vähentyä tai kasvaa, toisilla tieto lisää tuskaa, toisella tietämättömyys aiheuttaa ahdistusta. Yksilö siirtyy tilanteesta riippuen toiseen tilaan eikä suhtautuminen noudata ennalta-arvattavaa polkua. (Ponteva 2010, 42.)

Muutoksessa pärjäämisen keinot jaetaan ongelma- ja tunnekeskeisiin. Ongelmakeskeiset eli ratkaisukeskeiset keinot tarkoitus on vähentää muutoksen aiheuttamaa stressiä. Tunneperäisissä säädetään tunnereaktiota miellyttävämpään suuntaan. Ongelmakeskeiset keinot jaetaan viiteen ryhmään: 1) aktiivinen toiminta, eli toimitaan aktiivisesti tilanteen ratkaisemiseksi, 2) suunnittelu, 3) kilpailevien toimintojen karsiminen tarkoittaa joidenkin toimien vähentämistä, että pärjää muutoksessa, 4) maltti, eli henkilö hillitsee itsensä ja 5) välineellinen tuki, joka tarkoittaa neuvojen kysymistä, tilaisuuksiin osallistumista ja keskustelua. (Ponteva 2010, 55-56.)

Tunnekeskeiset keinot jakaantuvat yhdeksään ryhmään: 1) sosiaalinen tunnetuki, eli tunteiden jakaminen jolloin saadaan myötätuntoa ja ymmärrystä osaksi, 2) tunteisiin keskittyminen, jolloin tunteiden antaa purkautua, 3) tekemisestä luopuminen, 4) henkinen luopuminen jossa sijaistoiminnoilla viedään ajatukset muualle, 5) myönteinen uudelleentulkinta ja kasvu, 6) ongelman kieltäminen, 7) tilanteen hyväksyminen, 8) nautintoaineiden lisääntynyt käyttö ja 9) uskontoon ja henkisten asioiden puoleen kääntyminen. (Ponteva 2010, 55-56.)

Ymmärtämällä pärjäämiskeinot, organisaatio voi suunnitella tukemismallinsa, jotka voidaan edelleen jakaa tiedollisiin, osallistaviin, taloudellisiin ja psyykkisemotionaalisiin tukikeinoihin (Ponteva 2010, 68). Tiedollista tukea käyttöönotossa tarjotaan yhteisen koulutuksen ja organisaation sisäisen wikipedian kautta. Osallistaminen otettiin mukaan jo koulutusvaiheessa, kun starttipalaverit pidettiin hands on -työpajoina, joissa oma kone

muokattiin testausvalmiuteen ja jonka jälkeen jokaisella oli valmiudet testauksen aloitukseen. Tiedotusta tukivat myös yrityksen sisäiset keskusteluryhmät, jossa keskustellut aiheet voi kuka tahansa halutessaan lukea ja niihin kommentoida. Psyykkis-emotionaalinen tuki tarkoitti käyttöönottovaiheessa sitä, että henkilö nimettiin automaatiotestauksen eteenpäin viennissä, ja viestittiin että hänet sai koska tahansa kutsua paikalle keskustelemaan ja ratkaisemaan käyttöönotossa tulevia haasteita yksilökohtaisesti, tai pieninä ryhminä mikäli sama ongelma kosketti useampaa henkilöä. Turvallisuutta ja luottamusta kasvatettiin katselmointimallilla, jossa tehdyt käyttötapaukset ja automaatiotestit käydään läpi ja annetaan palautetta ja ideoita.

Yksilöt tarvitsevat ja mieltävät tuen eri tavalla. Usein yksilö selviää muutoksesta henkilökohtaisten keinojen kautta, jolloin organisaation tuen merkitys ei välttämättä ole suuri. Yleinen tiedotus, läpinäkyvyys ja tieto että tukea on tarvittaessa saatavilla luovat turvallisuuden ja vakauden tunnetta, jolloin usko muutoksen vahvistuu. (Ponteva 2010, 69.)

Viestintätapoja pohtiessa mietittiin sopivia viestintäväyliä. Vaikka sähköinen tiedotus on hyvä tapa, päädyttiin kutsumaan ihmiset yleisiin tiedotustilaisuuksiin, koska asiat selkenevät ihmisen selittäessä paremmin kuin kirjoitetusta tekstistä. Läpikäynnissä tarkennuksia ja selvennystä kysymyksiin voi kysyä heti. (Ponteva 2010, 70.)

Välittäminen on viestinnässä tärkeämpää kuin puitteet, joten tiedotustilaisuuden luonne ja keskustelu mahdollisuus on syytä miettiä. Esimerkiksi on todennäköisempää, että esimies, joka kutsuu tiimin taukotilaan pohtimaan asiaa kahvin ja suklaakeksin kera edistyy asian eteenpäin viemisessä paremmin kuin auditorioon ylimmän johdon virittämä Powerpoint -tilaisuus. Ero näissä on turvallisuuden tunne kommentoinnille.

Automaatiotestauksen eteenpäin viejät eivät olleet esimiestehtävissä, mutta toiminnassa noudatettiin seuraavia pelisääntöjä, jotka on johdettu Pontevan (2010) nimeämistä hyvistä esimiestaidoista. Ensimmäinen sääntö on omana itsenään olo, kertoen asioista mieluummin liikaa kuin liian vähän, kunhan pystyy myöntämään senkin, jos ei tiedä jotain. Toinen sääntö on puhumisen jatkaminen, kunnes viestisi on tavoittanut kaikki kuulijat. Kolmas sääntö on, että ajattelee ja viestii myönteisesti - kun on itse valmis muutokseen, asenne välittyy muihinkin. Myös vaikeista ja haasteellisista osioista voi etsiä hyvät puolet, ja kun virheitä löytyy ja etsitään, ei etsitä syyllistä. Sen sijaan korostetaan hyviä puolia ja viljellään hyvän tahtoista huumoria. Hymyillen asioihin on helpompi suhtautua innostuen. Tär-

keimmäksi tehtäväksi automaatiotestauksen eteenpäin viejille muodostui kuitenkin viimeinen sääntö: ole kannustava, ja madalla kynnystä aloittaa ja kysyä, sillä itse toimenpiteet työlle eivät olleet hankalia.

Asioiden selvittäminen ja toisten tukeminen ovat ennen kaikkea tahtoasioita, jotka vaativat peräänantamattomuutta. Lähtökohdan pitäisi olla, että apua saa aina kun sitä pyytää, sillä silloin avun pyytäminen on helppoa. Kun itse on saanut apua tarvitessaan, itsekin auttaa helpommin toisia. (Ponteva 2010, 103.) Tilanne on hyvä, jos ei voi nimetä ketään, joka ei olisi ollut valmis auttamaan, kun apua on tarvittu.

Tyypillisimmin käyttöönoton tuki koostuu suunnittelusta ja välineellisestä tuesta, mutta tarjolla on myös sosiaalista tunnetukea. Positiivinen asenne ja asioiden laittaminen oikeisiin mittasuhteisiin helpottavat myönteisiin asioihin keskittymistä ja uusien toimintamallien hyväksymistä. Kun työpaikalla keskustellaan asioista ja ihmissuhteet ovat kunnossa, muutoksessa pärjääminen on helpompaa. Keskustelumahdollisuuksien avaaminen arjessa ei ole pelkästään esimiehen asia, vaan siihen on jokaisella työntekijällä oma roolinsa. (Ponteva 2010, 58-59.)

3.3 Automaatiotestitapauksen elinkaari

Testauksen kannalta tavoitteet määritellään kahdella tavalla: 1) todennetaan, että toteutus täyttää sille asetetut vaatimukset ja 2) tehty toteutus on oikeanlainen ja ratkaisee ongelman, jonka vuoksi se tehtiin (Kasurinen 2013, 63). Esimerkiksi jos neuvotteluhuoneen varausominaisuudella voi varata neuvotteluhuoneen, mutta sovellus ei varmista onko päällekkäisiä varausta, sovellus voi täyttää annetut vaatimukset, mutta sen käytössä olisi haasteita.

Testitapaus on dokumentaatio vaatimuksista, toimenpiteistä ja odotetuista tuloksista, jotta voidaan varmistaa, että toiminnallisuus suorittaa toimet oikein (Myers, Sandler & Badgett 2012. 139). Testeillä pyritään ensimmäiseksi kattamaan kriittiset osuudet prioriteettijärjestyksessä, eli tärkeimmät ja virhealtteimmat toiminnot. Järjestelmä- ja regressiotestauksessa testitapaukset perustuvat sovelluksen aitoja käyttötilanteita jäljitteleviin skenaarioihin.

Jotta testitapauksen voi automatisoida, se täytyy pystyä ajamaan vaihe vaiheelta myös manuaalisesti. Syntaktinen virhe, eli koodausvirhe on helppo löytää tarkastamalla yksikkötestausvaiheessa erilaiset syötteet ja nykyaikaiset sovelluskehittimet osaavat niistä nostaa useimmiten myös virheilmoitukset. Haasteellisempi virhetyyppi on semanttiset virheet, joissa koodi toimii, mutta tekee vääriä asioita. Semanttisia virheitä ilmenee sovelluksissa, joissa on useita komponentteja ja liitoksia toisiin sovelluksiin. (Kasurinen 2013, 63.)

Teoriassa mikä tahansa testi voidaan automatisoida. Ydinkysymys on viekö testauksen automatisointi ja ylläpito enemmän rahaa kuin mitä se säästää. Testiä ei pitäisi automatisoida lainkaan, jos testi on tarkoitettu vain kerran ajettavaksi tai tulokset ovat vaikeita mitata ja varmistaa. Testauksen automatisointi kannattaa yleensä sellaisille testitapauksille, jotka täyttävät yhden tai useamman seuraavista kriteereistä: 1) stabiili, 2) regressiotesti, 3) savutesti, 4) korkean riskin ominaisuus, 5) datasidonnainen testi, 6) stressitesti, 7) moni-selaintesti tai 8) monilaite testi.

Testattavan ominaisuuden stabiilius kriteerinä perustuu siihen, että epävakaan ominaisuuden testien ylläpito voi vaatia merkittävästi ylläpitoa, jolloin toistojen tarjoamaa säästöä ei saada. Yksikkötestit voidaan tehdä jo kehityksen alkuvaiheessa, koska ne muuttuvat hyvin vähän ja vain tilanteissa, joissa määrittely muuttuu. Käyttötapausten kuvauksista osa voidaan kirjoittaa jo ennen kuin työ on aloitettu, mutta järjestelmätestit ja käyttöliittymätestit kannattaa tehdä vasta kun manuaalisissa testeissä ominaisuus on todettu toimivaksi.

Regressiotestit ovat testejä, jotka on aiemmissa sykleissä todettu toimiviksi, ja joilla varmistetaan, että kertaalleen korjattu ominaisuus ei hajoa uudelleen. Koska regressiotestejä ajetaan usein, ne ovat automatisoinnin prioriteettilistalla korkealla. Jos regressiotestejä on paljon, niitä ei kannata ajaa joka kerta kun sovellus käännetään. Savutestit ovat regressiotestien pienempi joukko, jotka on arvioitu niin kriittisiksi, että niiden epäonnistuessa muitakaan testejä ei kannata ajaa. Savutestit muodostavat testijoukon, joka jatkuvassa integraatiossa käännetään joka kerta kun sovellus käännetään. (Smart 2015, 50.)

Korkean riskin ominaisuus tarkoittaa kriittistä ominaisuuksia, jossa tapahtuvat virheet ovat kalliita. Esimerkiksi tallennuksessa tai rajapinnassa olevat virheet vaikuttavat tietojen oikeellisuuteen ovat kriittisiä, koska järjestelmään täytyy tehdä erillistoimenpiteitä, joilla tallentuneiden tietojen puutteet korjataan. (Smart 2015, 17.)

Datasidonnaisella testillä tarkoitetaan testejä, joissa keskeisintä on erilaisten syötteiden testaus. Esimerkiksi kirjautumisessa voidaan käyttää erilaisia käyttönimi ja salasana pareja. Stressitesti on datasidonnaisen testin variaatio, jossa tarkastelupisteenä on järjestelmän kuormittaminen useilla samanaikaisilla hauilla.

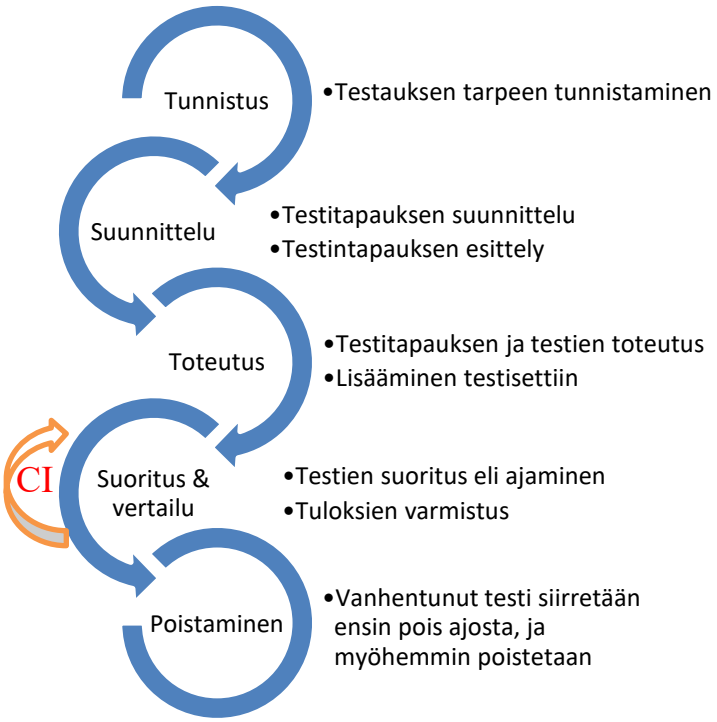
Moniselaintestien tarkoitus on varmistaa web sovelluksen toimivuus odotusten mukaan riippumatta selaimen versiosta. Tavallisesti ei ole tarpeen ajaa kaikkia mahdollisia testejä kaikilla laite-selainyhdistelmillä, vaan keskittyä tärkeisiin ominaisuuksiin ja suosituimpiin selaimiin. Eli asiakkaiden keskuudessa suosituimmalla selaimella ajetaan kaikki testit ja kriittiset tapaukset ainakin Chromella, Safarilla, Firefoxilla, Internet Explorerilla ja Microsoft Edge:llä. Monilaitetuessa testauksen keskiössä on mobiilituki ja suoriutuminen eri resoluutioista ja kokoisilla laitteilla.

Järjestelmätason automaatiotestit on jo aiemmin tehty järjestelmätestauksen yhteydessä, ja tästä on jäänyt testidokumentaatiota. Mikäli testihavaintojen ja kehittäjin ja testaajien arvioinnin mukaan automatisoinnille ei ole estettä, voi järjestelmätestin ohjelmoinnin aloittaa.

Testausprosessi lähtee liikkeelle automaatiotestauksen tarpeen tunnistamisesta, eli niiden ominaisuuksien ja toiminnallisuuksien löytämisestä joiden testaus halutaan automaatiotesteillä kattaa. Suunnittelu vaiheessa mietitään miten, millä työkaluilla ja millä tarkkuudella testit tehdään, ja mikä on niiden tarkoitus, eli mitä ne varmistavat. Toteutusvaiheessa luodaan testitapaukset, joista toteutetaan automatisoidut testit. Valmiit testit suoritetaan, minkä jälkeen muodostuu analyysi. (Fewster & Graham 1999, 13-17.)

Projektissa automaatiotesteistä tehdään testisuunnitelmat tikettien yhteyteen ja ne viedään versiohallintaan. Versionhallinta dokumentoi testin linkittymisen tiketeille ja työhön. Testistä voidaan jäljittää, kuka testin on tehnyt ja testin ylläpidon edetessä, miksi testiin on tehty muutoksia.

Kuviossa 7 on kuvattu automaatiotestin elinkaari. Kaaviossa on otettu mukaan jatkuva integraatio (Continuous Integration, CI). Jatkuvassa integraatiossa testisetteihin voidaan määritellä testit, joita ajetaan joka käännöksen (build) yhteydessä ja testit joita ajetaan versiojulkaisun yhteydessä. Tyypillisesti elinkaari päättyisi vertailu vaiheeseen, mutta elinkaareen lisättiin myös testin poistaminen.



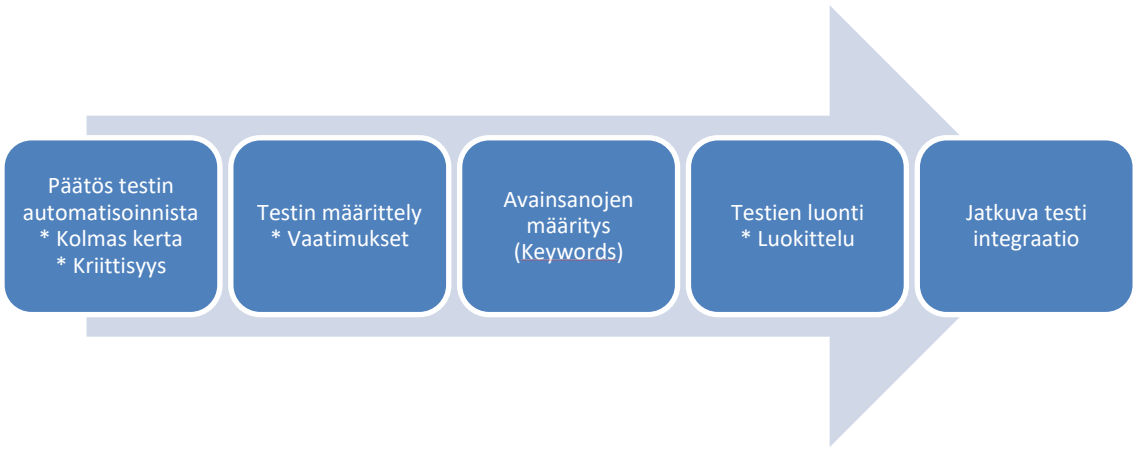
KUVIO 7 Automaatiotestin elinkaari

Testit luokitellaan ja sijoitellaan luokitusta vastaaviin testisetteihin. Mikäli testin suorituksessa tapahtuu virhe, tulosten tarkastaja siirtää tapauksen analysoitavien joukkoon. Analyysissa otetaan kantaa siihen, miksi testi ei toimi. Tyypillisiä syitä ovat virhe sovelluksessa, jolloin korjataan toteutusta, virhe testissä, jolloin korjataan testiä ja testiympäristön virhe, jolloin ympäristöolosuhteet aiheuttavat virheen. Jos toteutus muuttunut siten, että testin pitäisikin nostaa virhe, arvioidaan kannattaako testi korjata, vai onko sen testaama toiminnallisuus poistunut kokonaan käytöstä, jolloin se poistetaan käytöstä. Käytöstä poistetut tapaukset käydään läpi säännöllisesti.

Kuviossa 8 automatisoidun testitapauksen elinkaari käsitellään automaatiotestitapauksen ominaisuuksien kannalta. Yksikkötestitapaukset ovat itsenäisiä, joten niiden suhteen pohdintaa tarvitaan vähemmän, mutta integraatio- ja järjestelmätasolla testit ovat alttiimpia muutoksien vaatimille ylläpitotoimille.

Automaatiotestaus päätös on testien hallinnan kannalta tärkeä päätös. Kokemusten perusteella automaatiotestaustiimi suosittelee, että jos toiminnallisuutta tai kohdetta täytyy testata ainakin kolme kertaa alusta loppuun, se on mielekäs automatisoida toistokertojen puolesta. Käyttöönotossa todettiin, että ensimmäisen testin tekoon meni suhteessa enemmän, jopa kymmenkertainen aika, mutta ensimmäiset testit vievät suhteessa enemmän aikaa, kun tiimi on vasta oppimassa menetelmää ja pohjalla ei ole materiaalia. Vakiintuneemmassa tilassa testin automatisointi vei kolmesta viiteen kertaan kauemmin kuin pelkän testin ajo.

Testien luontivaiheessa testi luokitellaan sen vaativuuden ja tavoitteiden mukaan. Testaako testi kriittistä perustoimintoa, jonka tulee aina toimia? Silloin se on järkevä lisätä käännöstesteihin. Jos testi on tärkeä, mutta sen ajo on lähinnä hyväksymisluonteinen, se voidaan ajaa vasta testiin kääntövaiheessa, jolloin se jää kiinni ennen testausvaiheeseen siirtymistä, mutta ei hidasta jatkuvaa kääntämistä. Esimerkkejä tällaisista testeistä on harvinaisempien ominaisuuksien testit, joiden toimivuus halutaan varmistaa, mutta jotka eivät vaaranna normaaleja perusominaisuuksia. Suorituskyvyn havainnointi on testien luokittelussa tärkeää, sillä testien ajon ei tulisi hidastaa kehitystyötä.



KUVIO 8: Testitapauksen automatisointi prosessi tekijän toimien kannalta

Testin määrittely tehdään Azure Devops Services -ympäristössä testisuunnitelmiin, jotta testi kytkeytyy siihen liittyvään määrittelyyn. Tiketillä kuvataan mitä testi testaa. Avainsanojen määritys vaiheessa mietitään mitkä testin osista kannattaa muokata modulaarisesti uudelleenkäytettäväksi, ja mitä olemassa olevia rakenteita voitaisiin hyödyntää.

3.4 Työkalujen valinta

Automaatiotestaus työkalun valinnassa tärkein kriteeri on työkalun kyvykkyys automatisoida testejä oikeassa toimintaympäristössä, oli kyse tietystä käyttöliittymästä ja työpöytänäkymästä, web- tai mobiiliteknologiasta. Web-testauksessa työkalun kyky useiden selaimien testaukseen on tärkeää, kun taas mobiilitestauksessa voidaan tarvita oikeiden laitteiden simulointia ja emulointia.

Työkalujen pohdinnassa käytiin läpi seuraavia aihealueita:

- Käytetäänkö skriptauskieltä vai tehdäänkö testit samalla ohjelmistokielellä kuin kehitys?
- Tukeeko työkalu debuggausta ja refaktorointia?
- Miten automaatiota voidaan ajaa?
- Miten automaatio tunnistaa käyttöliittymän kohteet?
- Miten toiminnot käyttöliittymän kautta käsitellään?
- Lisensointi vaihtoehdot?
- Voidaanko objekteja ja työkaluja uudelleenkäyttää?
- Tarvitaanko tukea koodittoman automaation käytölle?
- Integroituuko työkalu muihin tarvittuihin työkaluihin ja kirjastoihin?
- Tukeeko työkalu versiohallintaa?
- Onko työkalu hyvä käytettävyydeltään?
- Tukeeko kehys valittua toimintatapaa (modulaarisuus, datalähtöisyys, käytöslähtöisyys, avainpohjainen, hybridi)?

Työkalujen valinnassa lähdettiin liikkeelle työkaluista, joista suunnittelutiimin jäsenillä oli jo aikaisempaa kokemusta. Tämän lisäksi kartoitettiin ja tutkittiin muiden työkalujen soveltuvuutta käyttöön.

Käännösympäristöksi valittiin lähtökohtaisesti myös versionhallintana toimiva Team Foundation Server, johon projektin aikana tehtiin migraatio Azure Devops Services -ympäristöksi. Perusteena tuotteelle on, että projektin tavoitteena ei ollut muuttaa jo käytössä olevia ja toimiviksi todettuja sovelluksia.

Työkaluksi valittiin yksikkötestauksessa xUnit, jolla on hyvä tuki kirjastoissa ja käytetyissä sovelluskehitysympäristössä. Tietojen mockaamiseen ja virtualisointiin valittiin FakeItEasy.

Ensimmäisessä vaiheessa käyttötapaus skenaarioita kirjoitettiin SpecFlow:lla. Testien tausta kirjoitettiin Selenium WebDriver kirjastojen kautta. Tässä vaiheessa testejä ajettiin kehityksen yhteydessä eikä kytkettynä jatkuvaan integraatioon.

SpecFlown suhteen haasteeksi ilmeni tuen hidas tulo, kun tuote on kirjoitettu .NET Core versiolla jolle SpecFlowssa ei projektin aikana ollut vielä tukea, eikä se tullut projektin aikana odotusten mukaan. Koska todettiin että tuote ei vastaa kehitystarpeita, skenaariot siirrettiin suoraan Robot Framework ympäristöön.

Käännöksen kannalta muuttui se, että suoran Selenium Webdriverin käyttämisen sijasta käytetään Robot Frameworkin SeleniumLibrarya. Ratkaisuissa ero on se että Robot Framework toimii ratkaisussa välikerroksena ja testien kirjoittaminen voidaan tehdä yksinkertaisilla python skripteillä.

Testauksen lisäarvo on varmistavan roolin lisäksi tuottaa nopeaa palautetta tiimille. Automatisoitu testaus mahdollistaa palautteen antamisen välittömästi muutoksen jälkeen. Testaus eivät kuitenkaan saa hidastaa tuotekehitystä. Siksi testejä on aktiivisesti hallittava ja monitorointi on tärkeä osa työtä. (Simbhoedatpanday 2015.)

Automatisoitu testaus kohdeorganisaatiossa tehdään loogisesti versioidussa ympäristössä, mikä tarkoittaa sovelluksen haarautumista useaan erin versioon. Kehityshaaroja on useita eri tuotteiden ja versioiden kehittämiseen. Automaatiotestauksen luokat ovat haarautettuja, sillä toisen haaran testit voivat toimia väärin tai olla toimimatta lainkaan koodipohjan erojen takia.

Automaatiotestauksen ensimmäinen toimenpide on varmistaa testausympäristön vakaus. Jos testiympäristö ei ole vakaa, testien ajossa voi nousta sellaisia virheitä, jotka eivät ole testitapauksen ajosta lähtöisin vaan ympäristön epävakaudesta. Esimerkiksi testi, joka testaa tapahtumien tallentumista sovelluksen kautta tietokantaan voi epäonnistua, jos tietokantapalvelin kovan kuormituksen alaisena. Mikäli virheiden kirjoittaminen lokiin on tehty oikein, virheen alkuperä saadaan helposti selville, mutta jos ei ole, voidaan aikaa

käyttää paljon virheen tutkimiseen tallennuspalvelusta, vaikka ongelma on testausympäristön tietokantapalvelimen liiallinen kuormitus.

Testauksen työkaluihin kuuluu kaikki testauksen aputyökalut aina hallintanäkymästä testitapausgeneraattoreihin, virheenjäljitystyökaluista testikattavuuslaskureihin ja kehitysalustoihin tuotaviin kirjastoihin ja aputyökaluihin asti. Testauksen työkaluissa käytiin läpi kaikki työkalut, joista oli kokemuksia, kartoitettiin muita vaihtoehtoja ja lopulta päätettiin ensimmäiset tuotteet, joilla testausta lähdetään suorittamaan. Kaikkia työkaluja ei otettu heti käyttöön, vaan niillä tehtiin kokeiluja ja pilotointeja niiden sopivuuden varmistamiseksi.

3.5 Proof of concept eli ratkaisun soveltuvuustestaus

Käytännön toteutuksessa automaatiotestauksen soveltuvuutta tuotteiden käyttöön testataan valmistelemalla listan testitapauksia, joiden automatisointi tehdään osana työkalun soveltuvuuden arviointia. Proof of concept -vaihetta voidaan kutsua myös demovaiheeksi sillä siinä valittujen työkalujen ja toimintatapojen toimivuus testataan käytännössä toimintaympäristössä.

Kuviossa 9 visualisoidaan Proof of concept-vaiheen eteneminen projektissa. Prioriteettilistasta valitaan avainskenaariot, joille tehdään tarkempi suunnittelu ja skenaariomäärittely, ja näiden testien taustalle luodaan geneerinen kirjasto. Vaiheen tavoitteena on tuottaa joukko valmiita testejä, jotka ovat luonteeltaan yleiskäytettäviä ja vaativat vähän ylläpitoa.

KUVIO 9 Proof of concept -vaiheen prosessi projektissa

Hyviä vaihtoja automaatiotestien ensimmäisiksi kohteiksi ovat savutestit, versionhyväksyntätestit ja korkean luokan regressiotestit. Käyttöönottoon valittiin regressiotestaus yhteen osa-alueeseen. Testeissä pitää olla vähintään yksi E2E (end-to-end) testitapaus, eli

testi, joka kattaa testialueen alusta loppuun. Tällä tavoin varmistetaan, että työkalut toimivat kaikissa avaintoimissa.

Proof of concept -vaiheeseen valittujen käyttötapaukset ovat valittuja käyttötapauksia laajalta skaalalta, mutta valinnassa huomioitiin, että liian isoa osaa toiminnoista ei testien kohteeksi kannata ottaa, sillä tämän projektivaiheen tarkoitus on työkalujen soveltuvuuden arviointi käyttöön, ja vaiheen kesto on verrattain lyhyt, vain muutamia viikkoja. Vaiheen tavoite on saada varmuus siitä, ovatko työkalut konseptiin ja toimintaympäristöön riittävät ja toimivat, jotta toimintoja kannattaa lähteä viemään pidemmälle ja kouluttamaan tiimeille.

Kun testitapaukset on valittu, ensimmäiseksi varmistetaan, että ne ovat paitsi hyvin dokumentoidut, myös hyvin suunnitellut. On tavallista, että manuaalitestitapauksiin täytyy lisätä tietoja, jotta ne ovat automatisoitavissa. Tyypillisiä täydennettäviä asioita ovat esiehdot, jotka testin suorittaminen edellyttää, ja se miten oikea toiminnallisuus testissä todennetaan. Manuaalitestauksessa testaaja osaa täyttää aukot, mutta automatisoinnissa määrittelyn tulee olla täydellistä. Yksikkötestitasolla testin nimeämis- ja kuvauskäytännöt voivat riittää dokumentaatioksi, mutta pidemmissä testeissä testitapauksen vaiheistuksen ja tapauksen kuvaus korostuu. Automaatiotestin kuvaukseen kuuluu lopuksi ajettavat puhdistus toimenpiteet, joilla tuloksien validoinnin jälkeen testitulokset puhdistetaan järjestelmästä.

Toinen keskeinen aihealue, jota manuaalitesteissä ei tyypillisesti ole määritelty on sääntöjen luonti virheiden käsittelylle. Virheiden käsittelyllä ei viitata tässä kohtaa negatiiviseen testaukseen, eli sen varmistamiseen, että sovellus toimii odotetusti myös virhetilanteissa. Säännöt tarvitaan siihen, miten automaatiotesti toimii, kun suorituksessa tulee odottamaton virhe, jotta ylläpitäjän on helppo paikantaa virhe ja sitä myöten arvioida millainen virhe on.

Kohdeorganisaatiossa testitapauksien kuvaukselle ja dokumentoinnille oli olemassa tavat, joiden pohjalta automaatiotestaustestitapauksen rakenne suunniteltiin. Automaatio testitapauksen rakennetta käsitellään laajemmin luvussa 3.3. Jos testitapauksille ei olisi ollut hyvää toimintamallia, olisi kannattanut luoda uusi pohja, johon testitapauksen esiehdot, testiaineisto, odotetut tulokset ja odotettu työpanos olisi määritelty.

4 AUTOMATISOITUJEN TESTIEN SUORITUS

4.1 Testien toteutus ja yhteiset pelisäännöt

Testien toteutukseen laadittiin yhteiset säännöt. Automaatiotestien arkkitehtuuri noudattaa tuotteen näkymärakennetta, ja nimeäminen on yhtenäinen tuotetason kanssa. Testauksen tueksi tehdään yhteisiä muuttujia ja apukirjastoja, jotka nopeuttavat toimintaa myöhemmin. Automaatiossa testiympäristöt rakennetaan Docker-kontteina, eli valmiiksi määritellyillä peruskuvilla, jossa on sovelluksen perustiedot ja riippuvat alustat asennettuna. Koska kontti on valmiiksi rakennettu, helposti siirrettävä ja stabiili, uudet ympäristöt voidaan muodostaa joka kerta uudelleen. Ratkaisun etu on, että kanta ja sovellus on aina samassa tilassa testiajon alkaessa ja tila on nopea palauttaa koska tahansa takaisin.

Yhteiset testiluokat suunnitellaan helposti muokattaviksi ja uudestaan käytettäviksi. Fyysisten resurssien osoitteet määritellään konfiguroitaviksi, jolloin ympäristö on helposti siirrettävissä ja monistettavissa. Päällekkäisten testien toteuttamista tulisi välttää, sillä päällekkäisyys ei anna lisäarvoa ja lisää testauksen ylläpitoa. Yhden testitapauksen tulisi testata vain tiettyä toiminnallisuutta testattavasta ohjelmasta.

Testien dokumentointi on tärkeää, jotta testien kanssa työskenteleminen on nopeaa ja käytännöllistä. Koska käytössä on skenaariopohjainen määrittely, testit dokumentoivat itse itsensä, joten yhteiset ohjeistukset koskevat nimeämistä ja arkkitehtuuria. Testitapauksen nimen on kuvattava mitä testi testaa. Raportoinnin helpottamiseksi työt linkitetään muutokseen, jossa siihen liittyvä toiminnallisuus on toteutettu. Usein on turhaa tehdä erillistä dokumentaatiota testitapauksien toiminnasta, koska testien riittävä kommentointi kertoo, mitä testataan (Loveland, Miller,Prewitt & Shannon 2004, 178).

Automaatiotestauksen toteutus laajentaa ohjelmointikäytäntöjä, sillä yhtenäiset käytännöt helpottavat koodin ylläpitoa, uudelleenkäyttöä, luettavuutta ja ymmärrettävyyttä. Ensimmäinen sovittu sääntö ja yleinen toimintatapa on testin nimeäminen sanalla testi – näin testi tunnistetaan missä tahansa yhteydessä siihen viittaus tuleekin. Toisekseen testit jaetaan moduulien mukaan omiksi testiluokikseen. Kolmanneksi jokaiselle testille määritellään kuvaus, joka kertoo mitä testataan ja mitä odotetaan lopputulokseksi. Lisäksi testistä pitää tulla ilmi testaako se negatiivisia vai positiivisia skenaarioita.

Jokainen automaatiotesti ajetaan useita kertoja läpi toteutusvaiheessa. Testin toimivuus varmistetaan katselmoinnissa, johon se lähetetään suorituksen jälkeen. Katselmoinnin läpäissyt testi voidaan siirtää integraatiopalvelimen testiajoihin.

Automaatiotestit ajetaan Azure DevOps Services -palvelimella osana jatkuvaa integraatiota. Automaatiotestauksen vakiintuessa käyttöön testejä tarvitaan myös erillinen palvelin, mutta siihen ei käyttöönottovaiheessa oteta kantaa, sillä käyttöönottovaiheeseen nykyiset ratkaisut soveltuvat hyvin.

Automaatiotestien ylläpito on merkittävä kuluerä, joten niiden ylläpidettävyys huomioidaan suunnittelussa alusta alkaen. Jos testiaineistossa käsitellään tuotannosta lähtöisin olevaa arkaluontoista tietoa kuten henkilötietoa, on testien tietojen käsittelyn syytä olla kryptattua, eli salattua, jotta testejä analysoivakaan henkilö ei saa käsiteltäviä henkilöitä selville. Aiheesta lisää pääluvussa 5 testidatan hallinta ja anonymisointi.

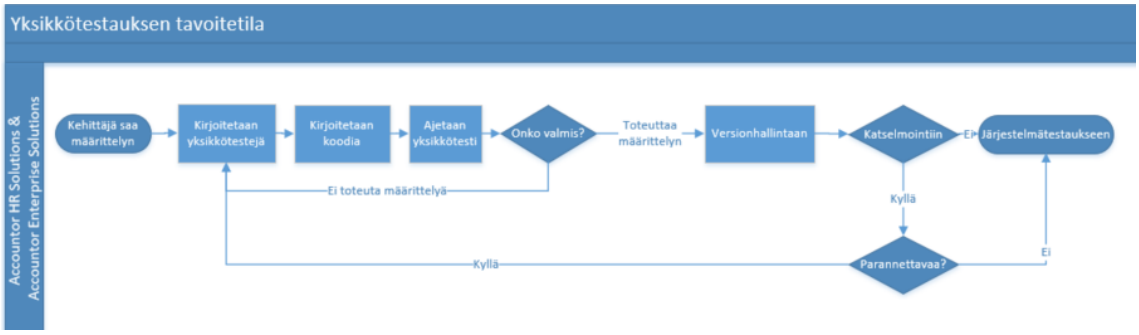
Hyvinkin toteutettu automaatiotesti vaatii elinkaarensa aikana ylläpitoa. Tästä syystä jokainen suurempi prosessi kannattaa pyrkiä toteuttamaan yleiskäyttöiseksi: kun prosessiin tulee muutoksia, korjattavia kohtia on vähemmän. Yhteiskäyttöiset prosessit sijoitetaan omiin näyttökohtaisiin luokkiinsa. Modulaarisuuden hyödyntäminen helpottaa uusien testien toteuttamista ja ylläpitoa, mutta ei vaadi juurikaan ylimääräistä työtä tekovaiheessa.

4.2 Testaustasot projektissa

Tuotteen laadun kannalta yksikkötestaus näyttelee merkittävää osaa, sillä on arvioitu että 70% bugeista pitäisi tulla yksikkötestausvaiheessa ilmi. Lisäksi testit muuttuvat harvoin, joten ne ovat edullisia ylläpitää. Yksikkötestauksen suunnittelussa lähdettiin ensin nykytilakartoituksesta liikkeelle, mutta koska automatisoitua testausta ei käytännössä tehty, ei nykytilallakaan ollut mitään määriteltyä prosessia.

Kuviossa 10 kuvataan testilähtöisen automaation mukainen prosessi, jossa kehittäjä kirjoittaa ensimmäiset yksikkötestit määrittelyn perusteella ennen kuin koodia aloitetaan kirjoittamaan. Näin yksikkötestejä voidaan ajaa ja täydentää koodin kirjoituksen aikana, ja

ne muodostuvat myös yhdeksi valmiin koodin laatukriteeriksi. Kun koodi suorittaa määrittelyn arvioidaan. onko tarvetta manuaaliselle katselmoinnille. Katselmointi vaihe ohitetaan vain, jos muutos että katselmointia ei katsota tarpeelliseksi.



KUVIO 10. Yksikkötestauksen tavoitetila

Yksikkötestauksessa yhden moduulin, funktion tai olion toimintaa testataan välittömästi toteutuksen yhteydessä, kehittäjän toimesta. Testin tavoite on varmistaa, että toteutettu toiminto tai muutos toimii ja reagoi annettuihin syötteisiin oikein. Usein komponentilla on sidoksia toisiin moduuleihin tai testidataan. Näissä tapauksessa testausta helpotetaan mock -komponenttien eli järjestelmän osia simuloivien testikomponenttien avulla. Simuloitujen komponenttien tavoitteena on varmistaa, ettei testeillä ole riippuvaisuuksia muihin testeihin. Itsenäinen testi voidaan ajaa aina samalla tavalla ja testaajalla on täysi hallinta siihen, mitä testi saa syötteeksi. Lisäksi testitapausten simuloinnilla voidaan helposti testata sellaisia virhetiloja, joiden aiheuttaminen on vaikeaa, esimerkiksi on helpompaa simuloida muistikortin kirjoitusvirhe, kuin pyrkiä rikkomaan muistikortti tilanteen aiheuttamiseksi. (Kasurinen 2013, 51-52.)

Testitapauksien laatimiseen muodostettiin tarkistuslista, joka löytyy liitteestä 1. Tarkistuslistassa käydään läpi yleisesti tunnistettuja tilanteita, joissa sovellus on altis virheille. Kasurinen (2013, 53) on tunnistanut seuraavat ongelmatilanteet:

- syötettyjen arvojen tyyppi,
- syötettyjen arvojen rajat,
- käyttäjän syöttämät arvot,
- toistorakenteiden rajat,
- muistinkäsittely,
- valintarakenteet,
- komponenttien rajat,

- käyttäjän valitsema järjestys tapahtumille,
- näkyvyysrajat,
- syntaksivirheet ja
- erikoismerkkien käsittely.

Sovelluskehitin ja apuohjelmat kuten Resharper auttavat tunnistamaan osan näistä virheistä sovelluksen kääntämisen aikana. Yksikkötestauksessa on tärkeää testata raja-arvot ja niiden ylitykset, sillä vaikka datan pituus on huomioitu jossain vaiheessa toteutusta, se ei välttämättä ole otettu huomioon kaikissa. Esimerkiksi on voitu asettaa, että kielikoodi kentän pituus on kannassa 2 merkkiä, mutta jos käyttöliittymä huolii pidemmän merkkijonon, tietoa voidaan menettää tallennusvaiheessa. Toiminnallisuuden testausta virheellisillä arvoilla ja poikkeustilanteissa kutsutaan negatiiviseksi testaukseksi, jonka tavoite on varmistaa sovelluksen oikea toiminta virhetilanteessa. Esimerkiksi merkkijonotyyppisillä tiedoilla on hyvä varmistaa, että ylimääräiset pilkut tai erotinmerkit tai erikoismerkit eivät riko käsittelyä.

Integraatiotestauksen tavoite on käytännön tasolla varmistaa, että osat toimivat yhdessä. Integraatiotestauksessa simuloituja sijaiskomponentteja, joilla korvataan osat, joita ei ole vielä tehty tai osat, jotka rajataan testauksen ulkopuolelle, kutsutaan nimellä tynkä (stub). Integrointijärjestyksiä on neljä:

1) Alhaalta ylöspäin, eli ensin testataan matalimman tason moduulit, kuten tietokannat ja verkkoyhteydet ja edetään seuraavalle tasolle, kunnes kaikki on testattu.
2) Ylhäältä alaspäin, eli aloitetaan järjestyksessä huipulta, esimerkiksi käyttöliittymästä ja laskeudutaan kohti rautatasoa.
3) Voileipätestauksessa integraatiota lähestytään molemmista suunnista yhtä aikaa.
4) Big bang, eli kertarysäystekniikka, jossa savutestin tapaan kaikki testataan kerralla. (Kasurinen 2013, 54-56.)

Järjestelmätestaus pitää sisällään kaiken mitä testataan koko järjestelmän käytössä ollessa. Tämä rajataan tässä dokumentissa pois, sillä järjestelmätestaus on käyttöönottovaiheessa manuaalitestauksena suoritettavaa – kantaa otetaan E2E-testaukseen, kuormitustestaukseen ja tutkivaan testaukseen, eli aihealueihin, joissa testauksen käyttöönottoprojektissa pyritään tuomaan muutoksia.

Järjestelmätestaus tehdään testiympäristössä ja sen tärkein ero hyväksymistestaukseen on, että sen aikana etsitään yhä virheitä yksittäisistä komponenteista. Hyväksymistestauksessa painopiste vaihtuu toiminnallisuuksien todentamiseen. (Kasurinen 2013, 57.) Hyväksyntätestausvaiheita voi olla myös useita, sisäinen version hyväksyntätestaus ja asiakkaan hyväksyntätestaus. Tietoturva huomioidaan kehittämisessä ja testauksessa koko ajan, mutta tietoturvatestaus on erillinen prosessi ja testauksen tyyppi, johon ei oteta kantaa tässä dokumentissa.

4.3 Määrittelystä integraatiotestaukseen Cucumberilla

Nopeasti kehittyvässä tuotealustassa määrittelyjen ja testien ylläpito on haaste, johon voidaan vastata yhdistämällä kehitys- ja liiketoiminnalliset osapuolet suunnittelussa ja ylläpidossa. Tuomalla testauksen jo määrittelyvaiheeseen mukaan, pystyttiin ottamaan askelia kohti käyttäytymislähtöistä kehitystapaa (BDD).

Lähtötilanteessa uuden ominaisuuden kuvaus lähti siitä, että tuoteomistaja kertoi tuoteasiantuntijalle, millaista ominaisuutta tarvitaan. Tuoteomistaja kirjoitti vaatimuksista kuvauksen, jonka kehittäjä luki ja muutti teknisiksi vaatimuksiksi. Jonka taas testaaja käänsi vaatimuksiksi, joita testasi. Testaaja myös kuvasi ominaisuuden sen valmistuttua asiakkaalle. (Smart 2015, 5.)

Tavoitetilanteessa tuoteomistaja keskustelee tuoteasiantuntijan kanssa siitä, millainen ominaisuus tarvitaan. Tuoteasiantuntija, kehittäjä ja testaaja käyvät vaatimukset yhdessä läpi ja määrittelevät vaatimukset käyttäjätarinoiksi, jotka kirjoitetaan yhtenäisessä formaatissa. Yhdessä vaaditut käyttäjätarinat ohjaavat kehittäjää toteutuksessa ja hän voi automatisoida testit kehityksen yhteydessä. Testaaja käyttää skenaarioita testien pohjana. Automatisoidut testit antavat palautetta kehitysprosessissa ja samalla dokumentoivat toteutetut ominaisuudet. (Smart 2015, 6.)

Epävarmuus toteutettavista ominaisuuksista on iso haaste sovelluskehityksessä, mutta raskas etukäteen kirjoitettu dokumentaatio ei toimi hyvin, kun on aika siirtää ymmärrys eteenpäin. Feature, eli toiminto on toimitettavissa oleva ominaisuus, joka ratkaisee jonkin

liiketoiminta tarpeen. (Smart 2015, 16.) Esimerkiksi tarve voisi olla tapahtumien siirtäminen toiseen järjestelmään tai houkutella asiakkaat käyttämään uutta ominaisuutta helpommalla ja kätevämmällä käyttöliittymällä.

BDD:ssä ei ole tarkoitus löytää kaikkia vaatimuksia kerralla, vaan keskustelu tuoteomistajan, kehittäjän ja testaajan välillä jatkuu kehityksen aikana, jolloin kaikilla on yhteisymmärrys siitä, millaisia ominaisuuksia toteutetaan. (Smart 2015, 17.)

Määrittelyssä käytetään määrittelysyntaksina Cucumberia, joka on Gherkin kieleen perustuva automaatiotestaustyökalu, jota käytetään integraatiotestaukseen ja hyväksyntätestaukseen. Gherkin on kuvauskieli, joka mahdollistaa käyttötapauksien kirjoittamisen ymmärrettävällä ja luettavalla kielellä. Käyttötapaukset käännetään niille luotujen vastinfunktioiden kautta koodikieleksi, joista ajettavat testit muodostuvat.

Käyttötapaus kuvaa testattavan sovelluksen ominaisuuksia se käytöksen näkökulmasta, ja ne voi koulutuksen ja läpikäynnin jälkeen kirjoittaa kuka vain. Suunnitteluvastuu on yhteinen tiimin jäsenillä, mutta testit kirjoittavat määrittelyiksi tyypillisesti testausroolin henkilöt. Käyttötapaus kuvaus menee ohjelmistokehittäjälle, joka määrittää askeleen vastinkappaleet järjestelmässä. näistä muodostuu testitapaus.

Feature-lohko testissä tarkoittaa yhtä toiminnallista ominaisuutta, esimerkiksi autentikaatio. Yhden ominaisuuden sisällä voi olla yksi tai useampi skenaariolohko, joissa on testiin kuuluvat askeleet, esimerkiksi autentikaation skenaarioita voisivat olla sisäänkirjautuminen, uloskirjautumien ja uudelleenkirjautuminen session vanhetessa. Jokaisella skenaariolla on askeleet. Askel alkaa aina Given avainsanalla, esimerkiksi jos määrittelisimme uloskirjauksen askelia, lähdemme lähtökohdasta, että käyttäjä on jo onnistuneesti kirjautunut sisään. When avainsanalla kerrotaan mitä tehdään, esimerkiksi käyttäjä painaa uloskirjautumispainiketta ja then avainsanalla mitä odotetaan, vaikkapa käyttäjä näkee tiedotuksen, jossa kerrotaan uloskirjautumisen onnistuneen ja kehotetaan sulkemaan selain välimuistin tyhjenemiseksi. Taulukkoon 1 on tiivistetty Gherkin kuvauskielen avainsanat

TAULUKKO 1 Gherkin kuvauskielen avainsanat

Avainsana	Selite
FEATURE	Sovelluksen ominaisuus, testin kohde
SCENARIO	Sovelluksen ominaisuuden käyttäytyminen
GIVEN	Asettaa testitapauksen lähtökohtaisen tilan ja parametrit
WHEN	Toiminnon toteutus tai määritys mikä toteuttaa ehdon
THEN	Odotettu toiminta ja ehdot, eli mitä pitäisi tapahtua
AND	Voidaan liittää useampia ominaisuuksia, toimintoja tai ehtoja joiden kaikkien tulee toteutua
OR	Voidaan liittää useampia ominaisuuksia, toimintoja tai ehtoja joista riittää että jokin toteutuu

Cucumberin heikkoudeksi käyttöönotossa ilmeni, että vaikka määritykset sai kirjoittaa luonnollisella kielellä, se koettiin monimutkaiseksi ja sopivia valmiita ja yleiskäyttöisiä kirjastoja ei ollut. joten liikkeelle lähtö oli hidasta. Kieliasu ja avainsanat täytyi sopia itse. Toisaalta yhden skenaarion pitäisi olla lyhyt pieni osa kokonaisuudesta, yksi määritelty ominaisuus, jotta testit olisivat helposti ylläpidettäviä ja luettavia.

Eduksi kielelle on, että se helpottaa kommunikointia, kirjoittamisen aloitukseen on pieni kynnys ja se dokumentoin itsessään testiympäristön toimintaa. Isot kokonaisuudet voidaan jakaa pienemmiksi käyttäjätarinoiksi, jotka on helpompi kehittää ja toteuttaa. Konkreettisten esimerkkien kautta toiminnosta on helppo keskustella. BDD ajatellaan usein käyttöliittymätason kuvaussyntaksiksi, mutta se on käytettävissä yhtä hyvin yksikkötestaustasolla.

4.4 Testien suoritus

Testit voidaan suorittaa joko manuaalisesti ajamalla ne ohjelmointiympäristöstä tai automaattisesti liittämällä ne osaksi ohjelman rakentamisprosessia. On tavallista, että testatut projektit voivat sisältävät yhtä paljon testiluokkia kuin ohjelman toiminnallisuuteen tarvittavia luokkia ja tietyt testit voivat olla hitaita suorittaa. Tämän takia jokaisella sovelluksen käännölle ei ajeta kaikkia testejä, vaan määritellään useampi testaustaso.

Suunnitteluvaiheessa vältetään pitkään kestäviä ja tehottomia testejä. Integraatio- ja yksikkötestit toteutetaan toisistaan erillään. Käyttöliittymätason testien suoritus on luonteeltaan hitain, joten tarvittaessa testejä tulisi pystyä ajamaan useassa ympäristössä yhtäaikaisesti. Kun eri testityypit ajetaan useassa eri sekvenssissä, rakennusprosessiin käytetty aika saadaan pidettyä pienenä.

Liittämällä automatisoidut testit rakentamisprosessiin saadaan tietoa mahdollisista virheistä nopeasti ja tehokkaasti. Lisäksi automatisoidut testitapaukset ajetaan aina kun sovelluksesta tehdään uusi versio, jotta pystytään varmentamaan ohjelman vanhojen ominaisuuksien toiminta (Loveland, Miller, Prewitt & Shannon 2004,170).

Tällaista jatkuvan testauksen ja käännön periaatetta kutsutaan jatkuvaksi integraatioksi (Continuos Integration, CI). Toimivan ohjelmiston rakentamisprosessin (build process), kuuluvat esimerkiksi lähdekoodien kääntäminen, tarvittavien tiedostojen kopiointi, tietokannan luonti ja yksikkötestien ajo. Kohdeorganisaatiossa ympäristö on olemassa ja toiminnassa, mutta sitä pystytään käyttämään tehokkaammin ottamalla mukaan automaatiotestausta tukevia toimia.

Projektin eri testaustasot kytkeytyvät version testauksen kypsyysmalliin. Ensimmäinen taso on automaattiset yksikkötestit, jotka ajetaan aina kun tehdään muutoksia koodiin ja käännetään koodi. Jos ne menevät lävitse, ajetaan integraatiotestit. Versiohallintaan viennin yhteydessä käynnistetään käännösprosessi, joka ajaa automaattiset hyväksyntätestit. Jos versio suoriutuu niistä voi alkaa tutkiva testaus. Lisäksi päivittäin voidaan ajaa kuormitustestit ja pidempiä testisettejä. Testaustasojen läpimenoaste muodostaa version kypsyysasteen.

Rakentamisprosessi osaa nykyisellään kääntää lähdekoodin ja raportoida käännön onnistumisesta, mutta sovellus julkaistaan erillisellä pyynnöllä, ja julkaisuversion tikettien merkintä on lähtötilanteessa puutteellinen. Tehostusta prosessiin saadaan, kun voidaan

- merkitä lähdekoodi käytettäväksi julkaisu versiossa
- ajaa yksikkö- ja integraatiotestit
- määritellä ajettava käännösputki, joka kääntää ohjelman testiympäristöön
- ajaa tietokanta päivitysskriptit
- ajaa suorituskykytestit ja toiminnalliset testit
- ilmoittaa sidosryhmille uusimmasta julkaisusta.

Raportin selkeys on tärkeää monitoroinnin helpottamiseksi. On voitava yhdellä silmäyksellä nähdä, menivätkö testit läpi, ja mikä niiden kattavuus on. Ongelmatilanteissa on tärkeää saada nopeasti informaatiota virheeseen menneestä testistä, jotta voidaan arvioida virheen vakavuutta ja vaadittavia toimenpiteitä.

4.5 Testitulosten raportointi ja ylläpito

Testaamisen tarkoitus on ylläpitää sovelluksen laatua, ja pääkeino siihen on virheiden havaitseminen. Virheestä sovelluksen toiminnassa eli poikkeamasta suhteessa määritykseen, käytetään tässä dokumentissa termiä bugi (Haikala & Mikkonen 2011, 205-206). Jos kunnollista määrittelyä ei ole, on oikeellisuuden todentaminen erittäin haasteellista ja perustuu täysin testaajan näkemykseen siitä mitä toteutuksessa on tavoiteltu.

Testi ajetaan ja tulosta tutkitaan suhteessa oletettuun tulokseen. Jos testi menee läpi, tarkastetaan odotetaanko sen onnistuvan vai ei. Yleensä ei odoteta virhettä. Toisaalta jos syynä virheoletukselle on ollut virhe tuotteessa, ja se on nyt korjattu, korjataan testi, ja testi itsessään hyväksytään onnistuneesti läpimenneeksi

Kun kyseessä on vielä toteuttamaton ominaisuus, negatiivinen testi tai tunnettu virhe, hyväksytään virheeseen meno, koska tulos on odotettu. Automaatiotesti epäonnistuu vain silloin, kun vastaus poikkeaa odotetusta tuloksesta. Toisinaan testi voi olla jonkin aikaa myös tiedostetusti virhetilassa, eli epäonnistuva, esimerkiksi silloin kun testattavassa toiminnallisuudessa on tiedossa bugi. Tällainen tilanne tulisi olla selkeästi testiin merkitty, ja tila tulisi päivittää, kun virhe on korjattu. Tähän on sovittava raportointiin ja analysointiin toimintatavat.

Yksinkertaisimmillaan testeistä halutaan raportoida status eli onko testien suoritus valmis ja menivätkö ne läpi vai eivät. Testien epäonnistumisen kriteerit vaihtelevat, tyypillisiä varmistuksia ovat saadun tuloksen vertaaminen odotettuun tulokseen. Läpimenneistä testeistä ei tavallisesti tarvita enempää tietoa, mutta epäonnistuneesta testistä tarvitaan yksityiskohtia ongelmasta, jotta voidaan päätellä miksi testin ajo ei onnistunut.

Testausraportista pitää ilmetä

- testatun järjestelmän nimi ja versio
- lyhyt tiivistelmä ajetuista testeistä ja niiden statuksista

- läpimenneiden ja suoritettujen testien yhteismäärä ja
- lista virheistä.

Työkalun ominaisuuksien mukaan raportti voidaan luoda joko testin yhteydessä tai jälkikäteen. Automaatiotestin suorituksen epäonnistuessa tarvitaan tapauskohtainen analyysi, jossa selvitetään epäonnistumisen syy. Onko kyse bugista, virheestä testissä vai oliko testiympäristössä suorituksen aikana testien suorituksen estävä häiriö. Tähän tarvitaan yleensä testauslokia. Testauslokin tarkkuus on tavallinen haaste, jos lokiin kirjataan liian vähän, ei tieto riitä ongelman ratkaisemiseen, ja liian paljosta oikean tiedon hakeminen on hidasta. Testien korjaamisesta lähetetään sidosryhmille tieto tilanteen normalisoitumisesta. Lisäksi korjaus katselmoidaan.

Virheet sovelluksessa priorisoidaan, koska on päätettävä estääkö virheen ilmeneminen esimerkiksi version julkaisun. Virheen priorisointiin vaikuttaa kaksi asiaa vakavuus ja ilmenemistodennäköisyys. Virheen vakavuutta arvioidaan sen vaikutuksen perusteella – onko kyse käytön estävästä virheestä vai pienestä kosmeettisesta puutteesta (Haikala & Mikkonen 2011, 205-206). Ilmenemistodennäköisyys arvioidaan sen perusteella kuinka suurta käyttäjäjoukkoa ongelma koskettaa.

Testituloksista raportointi tavat ovat merkittävät, sillä tuoteomistaja tarvitsee riittävän tarkan kuvauksen ongelmasta, jotta voi arvioida ongelman vakavuuden sovellukselle. Luokittelutapoja on monia, joten ongelman vakavuutta päätettiin arvioida kolmiportaisella asteikolla 1) pieni, luonteeltaan kosmeettinen ei estä sovelluksen käyttöä, 2) normaali jolloin ongelma haittaa käyttöä, mutta ei estä avainskenaarion toteutusta ja 3) suuri ongelma estää ohjelman käytön. Automaatiotesteissä havaitusta virheestä luodaan paitsi korjaustiketti myös kirjaus itse testiin, jotta kaikki näkevät, että virhetilanne on havaittu.

Kun ohjelmistosta julkaistaan uusi versio, tarvitaan aina uusia testejä ja vanhojen testien muuttamista toimimaan uuden ohjelmiston mukaisesti. Osa vanhoista testeistä saattaa tulla tarpeettomiksi, koska niiden testaama toiminnallisuus puuttuu kokonaan uudesta versiosta tai uudet testit korvaavat niiden toiminnallisuuden. (Fewster & Graham 1999, 191.)

Testien ylläpitoon vaikuttaa moni asia kuten,

- testien määrä

- testien käyttämän tiedon määrä
- testien ajamiseen kuluva aika
- testien dokumentaatio
- testien monimutkaisuus.

Mitä enemmän on testejä, sitä enemmän kuluu aikaa ja vaivaa testien ylläpitämiseen. Jos testit käyttävät ulkopuolista tietolähdettä, tarvitsee myös kyseiset tiedot päivittää ohjelmaa ylläpidettäessä. Testien suorittamiseen kuluva aika vaikuttaa myös testien ylläpidon kestoon, koska vikojen etsintä pitkään kestävistä testeistä on hitaampaa. Pitkäkestoiset testit voivat myös testata useampaa asiaa samassa testissä, jolloin yhden asian korjaaminen testistä ei välttämättä korjaa koko testiä. Testien ylläpito vaikeutuu huomattavasti, jos testien dokumentaatio ei ole riittävä. Testit tulisi pitää mahdollisimman yksinkertaisina, koska monimutkaisten testien ymmärtämiseen ja ylläpitämiseen kuluu paljon aikaa. (Fewster & Graham 1999,192-198.)

4.6 Regressiotestaus

Regressiotestauksella pyritään varmistamaan, että sovellus toimii oikein ja määritysten mukaisesti vielä sovellukseen kohdistuvien muutosten jälkeen. Regressio, eli muutoksesta johtuva virhe, voi aiheutua esimerkiksi uusien toiminnallisuuksien toteuttamisesta tai virhekorjausten tekemisestä. Kattavalla regressiotestauksella havaitaan muutoksen aiheuttamat odottamattomat ongelmat vanhoissa sovelluksen osissa – myös muutetun komponentin ulkoisilla osa-alueilla. Mitä laajemmasta sovelluksesta on kyse, sitä todennäköisemmin regressioita syntyy jossain ohjelman osassa. Tästä syystä regressiotestaus on hyvin tärkeää sovelluksen eheyden ja toimivuuden säilyttämiseksi. (Myers, Sandler & Badgett 2012. 134, 162.)

Kohdeorganisaatiossa tuotekehitys on muuttanut luonnettaan kohti iteratiivista ja ketterää. Iteratiiviselle sovelluskehitykselle on ominaista nopeat ja suuret muutokset, jolloin regressiotestauksen merkitys korostuu. Sovellus on muutoksen yhteydessä usein alttiimpi toteutuksesta johtuville virheille kuin uusi tyhjästä toteutettava sovellus (Myers, Sandler & Badgett 2012. 22).

Regressiotestit tulisi suorittaa määräajoin ja sovelluksen laadun varmistamiseksi mahdollisimman usein. Ideaalitapauksessa kaikki regressiotestit ajettaisiin jokaisen pienimmänkin muutoksen jälkeen, mutta käytännössä tämä on testien määrän ja kattavuuden kehittyessä hyvin vaikeaa tai tietyissä tapauksissa mahdotonta, sillä kattava regressiotestaaminen vie runsaasti aikaa. Tästä syystä regressiotestit on hyvä yhdistää jatkuvaan integraatioon ja versiokääntöön, jolloin ne ajetaan aina kun versiosta halutaan kääntää. uusi versio.

Regressiotestaus automatisoinnille hyvä kohde, sillä automaatio tuo regressiotestaamiseen useita hyötyjä: työmäärä testien ajamiseksi on pienempi, testejä voidaan ajaa nopeammin ja useammin. Automaation ajamat testit toistetaan aina samalla tavalla ja automaatiolla voidaan toteuttaa testejä, jotka olisivat manuaalitestauksella vaikeita tai jopa mahdottomia toteuttaa. Tällaisia voivat olla esimerkiksi testit, jotka vaativat epäinhimillisen nopean suoritusnopeuden tai testit, jotka seuraavat reaaliajassa taustajärjestelmien toimintaa. (Fewster & Graham, 1999, 6.)

Automaatiotestien tärkeimpiin ominaisuuksiin kuuluu uudelleenkäytettävyys. Kerran toteutetut automatisoidut regressiotestit voidaan ajaa sellaisenaan koska tahansa uudelleen. Samoja automaatiotestejä voi käyttää esimerkiksi useiden sovellusversioiden samanaikaiseen testaamiseen; monta asiakasta käsittävässä projektissa voidaan samat automaatiotestit suorittaa erikseen jokaiselle asiakkaalle. Regressiotestiä ei ole mielekäs automatisoida, mikäli sitä ei ole tarpeen suorittaa useita kertoja. (Fewster & Graham, 1999, 497.)

5 TESTIAINEISTON HALLINTA JA ANOMYSOINTI

5.1 Testiaineiston muodostus

Aineistoa testaukseen saadaan kahdesta lähteestä: joko itse muodostamalla, tai tuotannosta. Lisäksi tietoja voidaan kopioida ja järjestää testauksen tarpeiden mukaan esivalmistellusta malliaineistosta. Testiaineiston luominen tyhjästä manuaalisesti on hidasta. Tilannetta parantaa, jos testiaineiston luontiin on käytettävissä generaattoreita, tai sellaisia voidaan luoda. Etuna itsetuotetussa testidatassa on, että sen sisältö ja historia tunnetaan täysin. Mutta ympäristössä, jossa tieto on dynaamista ja asiakaskohtaisesti räätälöitävää, testaajan testit eivät välttämättä tekovaiheessa tunnista ja tiedä kaikkia skenaarioita, joita asiakas todellisuudessa käyttää – tai tulee myöhemmin toiminnan jatkuessa käyttämään.

Tuotantodatassa tiedot eivät aina välttämättä ole ihannetapauksen kaltaisia – tietoja voi puuttua, ne voivat olla väärin, ja niitä saatetaan täydentää eri aikoihin. Kolmannen osapuolen sovellukset saattavat tuottaa dataa, jonka muotoon tulee muutoksia sovelluksen päivittyessä. Tuotantodata antaa testauksessa tietoa sovelluksen todellisesta vakaudesta ja virheherkkyydestä, mutta testiaineiston turvallisuuden varmistus on huomioitava läpi prosessin. (Venkataramanan & Shriram 2017, 127.)

Tuotannosta lähtöisin olevan datan vaikeus on, että se on usein liian iso, hidas ja kallis ylläpidettäväksi testauskäytössä. Tuotantodatan turvallisuutta kasvattavat toimet ratkaisevat osittain kyseisen datan ongelmia, mutta suojauksilla on aina myös haittapuolia. Luvussa 5.6. käydään läpi datakeskeisiä anonymisointikeinoja, joista osajoukon valitseminen ratkaisee ison tuotantodatan käsittelyongelmia. Mutta sopivan otannan valitseminen on hidasta, ja testikattavuus ei ole yhtä hyvä kuin koko datalla. (Heimola 2017.)

Tuotteen kehittyessä testaus ei aina vaadi uuden testausdatan hakua, mutta datan tietyt ominaisuudet saattavat muuttua tai tietoja saatetaan tarvita enemmän uusien ominaisuuksien testaamiseen (Venkataramanan & Shriram 2017, 128). Koska anonymisoitu tieto ei ole enää palautettavissa takaisin tiettyyn henkilöön tai projektiin, testausaineisto vanhenee. Vanhentuneella aineistolla tehtyjen testien tulokset voivat olla epäluotettavia (Heimola 2017).

Tuotantodatan hallinnalle on määriteltävä prosessi, jossa määritellään milloin ja kenen toimesta ja mihin tarkoitukseen testausdataa voidaan tuotannosta ottaa. Testiaineiston käsittelyoikeuksien on oltava rajatut ja dokumentoidut. Testauksessa on syytä käyttää synteettistä dataa, jos vain mahdollista. Synteettisellä testausaineistolla tarkoitetaan anonymisoitua tuotantodataa, josta eriytetään testiin tarvittava osajoukko. Testausaineiston dokumentointi on tärkeää, jotta suojattu data sisältää testauksen kannalta relevanttia tietoa, mutta tiedon turvallisuutta uhkaamatta. (Heimola 2017.)

Testiaineiston tietojen laadulla on kriittinen rooli sovelluksen toimivuuden varmistamisessa. Keskeisin haaste on testiaineiston luonti ja ylläpito. Jos testiaineisto muodostetaan käyttämällä paljon generoitua tietoa, tulee ylläpidosta raskasta, koska yksittäisen testin suhteen on vaikea päätellä mikä osa datasta on testausta varten oleellista ja mikä ei. Toisaalta jos aineistoon tehdään muutoksia, korjattavia paikkoja on paljon.

5.2 Testiaineiston dokumentointi

Testiaineiston dokumentointi on tärkeää aineiston käytön kannalta, ja korostuu kun aineisto on peräisin tuotannosta. Testidatan hallinta ei ole tekninen ongelma, vaan pikemminkin strategian asettamista sille, mitä vaiheita ja testitapauksia automatisoidaan. (Heimola 2017).

Dokumentoinnille luotiin ohjeistus, joka kuvaa testiaineiston käsittelyn. Se säilytetään testiaineistosta erillään turvallisessa sijainnissa. Ohjeistus alkaa aineiston kuvauksella, jossa nimetään aineiston käyttötarkoitus. Kun aineistoa ei tarvita enää alkuperäiseen tarkoitukseen, se hävitetään asianmukaisesti. Dokumentissa käsitellään vastuut testidatan säilytyksen suhteen.

Jos dokumentointia ei ole tehty, aineistoa on hankala käyttää ja ymmärtää missä tilanteissa dataa voi hyödyntää. On määriteltävä käyttötarkoitus ja tarve, ja tunnettava data. Tämän jälkeen määritellään prosessit, joilla riskejä saadaan minimoitua käyttötarkoituksen puitteissa. Ja lisäksi on määriteltävä, miten datasta kommunikoidaan, miten se suojataan ja mitä tehdään tilanteissa, joissa dataa on jaettava tai siirrettävä jonkun tahon kanssa.

Viimeisenä on suunniteltava riskienhallinta, eli mitä tehdään, jos aineisto joutuu kolmannen osapuolen käsiteltäväksi, tulee jaetuksi epäturvalliseen sijaintiin tai muulla tavalla vaarantaa arkaluontoisia tietoja. (Elliot, Mackey, O´Hara & Tudor 2016, 3.)

5.3 Testiaineiston anonymisoinnin tarve

Testauksen laatu riippuu testimateriaalin laadusta ja kattavuudesta. Sovelluksen toiminnan ollessa dynaamista ja vahvasti datasidonnaista korostuu testauksessa tarve mahdollisimman aidon rakenteisen testiaineiston käyttöön. Kaikkia mahdollisia skenaarioita ei ole mielekästä testata, sillä korkeimmalla prioriteetilla on asiakkaan käyttötapojen ja tietorakenteiden toimivuuden varmistus.

Testauksen tavoitteena on löytää ongelmat sovelluksesta. Testaukseen saatetaan kopioida dataa suoraan asiakkaan järjestelmästä, jolloin voidaan varmistua sovelluksen toimivuudesta juuri kyseisen ympäristön tietorakenteilla. Tyypillisiä syitä haluta käyttää oikeaa tuotantodataa tai sen kopiota ovat aikataulu ja resurssit, joita varmistustoimenpiteille on asetettu. Jos ei ole mahdollista tehdä manuaalisesti riittävän laajaa testiaineistoa, tai aineiston tekemiseen ei ole resursseja, tuotantodata voi tarjota nopeasti aineiston. Manuaalisesti tuotettu data myös tuotetaan tyypillisesti tuntien millaista datan tulisi olla, ja asiakkaan oman tietorakenteen erityispiirteet jäävät testauksen ulkopuolelle.

Uuden testiaineiston kannalta on tärkeää saada mahdollisimman oikeantyyppinen esimerkki siitä, millainen oikea data voisi olla. Testikattavuus on tärkein testauksen tavoite, sillä satunnaiset testit eivät anna kunnollista tietoa koko sovelluksen toimivuudesta. Datan anonymisointi ja suojaus heikentää jossain määrin testien kattavuutta ja käytettävyyttä. (Venkataramanan & Shriram 2017, 129.)

Tietojen anonymisointia tarkastellaan teknisten toimien ja kontekstin kautta. Data voi olla arkaluonteista tai salaiseksi luokiteltua monesta syystä, mutta henkilöön liittyessään dataa tulee käsitellä erityistä varovaisuutta noudattaen. Teknisesti mietitään, kuinka todennäköisesti henkilö voidaan anonymisoidusta datasta tunnistaa ja miten hallitaan riskiä. Kontekstuaalisesta näkökulmasta tunnistetaan tekijöitä, jotka vaikuttava tunnistusriskiin, huomioiden datan elinkaaren, lailliset ja eettiset vastuut sekä ohjeistukset ja suositukset. Osa

suunnittelua on vastuiden ja toimenpiteiden määrittely, jos kuitenkin käy niin että aineistot ovat vaarantuneet tai päässeet vääriin käsiin. (Elliot, Mackey, O´Hara & Tudor 2016, 2.)

Anonymisointia tehdessä tulee huomioida tietolähteisiin ja sijainteihin liittyvä metadata, joka voisi johtaa henkilötietoihin. Tällaista tietoa voisi olla esimerkiksi laitteen omistajan tiedot, sijaintitiedot ja käyttäjäprofiilin liittyvät tiedot.

Aineiston anonymisointiin ei ole olemassa valmista kaikkiin aineistoihin soveltuvaa menettelytapaa, sillä toimenpiteet riippuvat käyttötarkoituksesta ja datan laajuudesta. Anonymisointi tulee suunnitella aina aineistokohtaisesti ottaen huomioon aineiston ominaisuudet, käyttöympäristö ja käytettävyys. Lisäksi menetelmät ja suojauksen kattavuus tulee katselmoida säännöllisesti, jotta varmistutaan että suojaus on riittävä.

5.4 Henkilötiedon erityisvaatimukset testiaineistossa

Henkilötieto käsitellään aina erityisenä tietona. EU:n tietosuoja-asetus määrittelee anonyymin tiedon toiminnallisesti. Jotta voidaan määrittää, onko luonnollinen henkilö tunnistettavissa, on huomioitava kaikki keinot, joita joko rekisterinpitäjä tai muu henkilö voi kohtuullisen todennäköisesti käyttää mainitun luonnollisen henkilön tunnistamiseen suoraan tai välillisesti. (EU 2016/679, 2016, johdantolause (26).)

Tärkeimmät muutokset aineiston käsittelyn kannalta ovat henkilötiedon käsitteen laajeneminen ja yksilön kasvaneet oikeudet omiin tietoihinsa. Henkilöllä on oikeus saada selonteko itseensä liittyvistä talletetuista tiedoista. Lisäksi hänellä on oikeus tulla unohdetuksi, eli tieto pitää pyynnöstä poistaa järjestelmästä ja peruuttaa myös suostumus käsittelyyn. (Heimola 2017.) Poikkeuksena on tieto, jonka käsittelyyn on laillinen peruste unionin oikeuteen tai jäsenvaltion lainsäädäntöön perustuen, tai jos käsittely tapahtuu yleistä etua koskevan tehtävän suorittamiseen, kuten kansanterveyden etua koskeviin töihin. Tieto voidaan olla poistamatta myös, jos tiedolla on yleisen edun mukaisia arkistointitarkoituksia, se on mukana oikeuskäsittelyssä tai käsittely on tarpeen sananvapauden tai tiedonvälityksen vapautta koskevan oikeuden käyttämiseksi.

Luonnollisen henkilön kohtuullisen tunnistamisriskin arvioinnissa huomioidaan kaikki objektiiviset tekijät, kuten resurssien tarve sekä käsittelyajankohtana käytettävissä oleva

teknologia ja tekninen kehitys. Koska teknologia kehittyy ja toissijaisia tietoja on saatavilla paljon muun muassa sosiaalisen median kautta, tunnistuksen todennäköisyyttä on arvioitava säännöllisesti.

Tunnisteellista aineistoa voi käyttää testaukseen silloin, kun se on tarkoituksenmukaista, suunniteltua, asiallisesti perusteltua ja tietojen käsittelyyn on laillinen käsittelyperuste. Tutkittavien näkökulmasta käsittely muodostaa riskin.

Tunnisteellisen aineiston turvallinen käsittely edellyttää suunnitelmaa, jossa huomioidaan paitsi itse dataan kohdistuvat turvatoimet, myös siirtoon ja säilyttämiseen liittyvien toimien tietoturvan suunnittelu ja monitorointi. Henkilötietojen käsittelyyn sovellettavista suojatoimista tyypillisimmät ovat minimointi, pseudonymisointi ja anonymisointi. Minimointi datan käsittelyssä tarkoittaa sitä, että käsittelyyn otetaan vain käyttötarkoituksen kannalta tärkeää tietoa - muu poistetaan datasta. Pseudonymisoinnissa tiedosta kooditetaan vahvat tunnisteet. Anonymisointi tarkoittaa datan käsittelyä tunnisteettomaksi, ettei sitä voida yhdistää enää luonnolliseen henkilöön. Anonymisointi on välttämätön, jos aineistoa halutaan pitkäaikaiseen tai uudelleenkäytettäväksi. (Venkataramanan & Shriram 2017, 22-23.)

Henkilötietojen osalta tietojen käsittelyä säädetään lailla ja säädöksillä. EU:n tietosuoja-asetuksen määritelmän mukaan henkilötiedoilla tarkoitetaan kaikkia tunnistettuun tai tunnistettavissa olevaan luonnolliseen henkilöön liittyviä tietoja. Data on tunnistettavissa, mikäli siihen liittyvä luonnollinen henkilö voidaan suoraan tai epäsuorasti tunnistaa tunnistetietojen, kuten nimen, henkilötunnuksen, sijaintitiedon, verkkotunnistetietojen taikka yhden tai useamman hänelle tunnusomaisen fyysisen, fysiologisen, geneettisen, psyykkisen, taloudellisen, kulttuurillisen tai sosiaalisen tekijän perusteella. (EU:n tietosuoja-asetus, artikla 4, kohta 1.) Määritelmän mukaisesti henkilötiedot aineistoissa eivät rajaudu vain tutkittavia koskeviin tietoihin, sillä tietoihin voi sisältyä tunnistetietoja kolmansista osapuolista. Myös heitä tunnistettavasti käsittelevät tiedot ovat aina henkilötietoja. (Elliot, Mackey, O´Hara & Tudor 2016, 23.)

Henkilötiedon sisältöä tai luonnetta ei ole rajattu vaan kaikki luonnolliseen henkilöön liittyvä tieto voi olla henkilötietoa, oli kyse perhesuhteista, terveydentilasta, fyysisistä ominaisuuksista, taloustiedoista, sosiaalisesta käyttäytymisestä, ammatillisesta toiminnasta, mielipiteistä tai väitteistä. Tiedon turvaamisen kannalta tiedon oikeellisuudella ja

todennettavuudella ei ole merkitystä. Testauksen näkökulmasta todellista muuntamatonta tietoa kannattaa säästää mahdollisimman vähän – esimerkiksi syntymäpäivät ovat merkityksellisiä harvoissa testeissä, ja voidaan korvata useissa tapauksissa luokitellun jakson keksityllä muuttujalla.

5.5 Henkilömuotoisen tiedon luokittelu

Tiedon anonymisoinnissa ei riitä suorien ja vahvojen epäsuorien tunnisteiden poistaminen, sillä mitä enemmän tietoa ja mitä yksityiskohtaisempaa se on, sitä helpompi tiedoista on henkilö tunnistaa. Siksi suorien tunnisteiden poiston jälkeen analysoidaan epäsuorat tunnisteet aineistosta ja arvioidaan voiko henkilön niiden perusteella tunnistaa.

Ensimmäinen vaihe aineiston läpikäynnissä on luokitella tiedot neljään luokkaan: suorat tunnisteet, epäsuorat tunnisteet, arkaluonteinen data ja yleinen, ei arkaluonteinen data (Venkataramanan & Shriram 2017, 131). Liitteessä 2 on muistilista suorista tunnisteista, eli tiedoista, jotka yksin riittävät tunnistamaan henkilön. Suoriksi tunnisteiksi tunnistettiin esimerkiksi henkilön koko nimi, henkilönimen mukainen sähköpostiosoite, henkilötunnus, passin numero ja biometriset tunnisteet kuten sormenjälki, kasvokuva ja käsin tehty allekirjoitus. Suorat tunnisteet riittävät yksistään henkilön tunnistamiseen, joten ne poistetaan tai korvataan testiaineistossa aina. (Venkataramanan & Shriram 2017, 132.)

Epäsuora tunniste on tieto, joka ei yksin riitä tunnistamiseen, mutta yhdistettynä muihin tietoihin voi mahdollistaa henkilön tunnistamisen. Tyypillisiä epäsuoria tunnisteita ovat sukupuoli, ikä, koulutus, ammattiasema, tehtävä tai toimi, tulot, siviilisääty, kieli, kansallisuus, etninen tausta, työpaikka tai kotikunta. Vahvoiksi tunnisteiksi kutsutaan sellaisia epäsuoria tunnisteita, joiden perusteella toisen aineiston kautta henkilö voidaan tunnistaa yksiselitteisesti. Tyypillisiä esimerkkejä vahvoista epäsuorista tunnisteista ovat postiosoite, järjestelmän käyttämät yksilölliset tunnisteet, puhelinnumero, auton rekisteri, tilinumero, tietokoneen IP-osoite, muu kuin henkilönimenmukainen sähköpostiosoite ja henkilöstä tunnistetietoja sisältävän verkkosivuston osoite, harvinainen ammattinimike, hyvin harvinainen sairaus tai vain yhdelle kerrallaan annettu asema. Myös harvinainen tapahtuma voi olla avain henkilön tunnistamiseen. (Venkataramanan & Shriram 2017, 132-133.)

Päivämäärä voi olla epäsuora tunniste, mistä tyypillisin esimerkki on syntymäaika, mutta myös kuolinpäivämäärät tai uutiskynnyksen ylittäneet tapahtumapäivämäärät ja vastaavat voivat olla epäsuoria tunnisteita yhdistettynä muihin tietoihin. Esimerkiksi työsuhteen aloituspäivä voi johtaa henkilön identifiointiin, jos hän on sosiaalisessa mediassa viestinyt asiasta. (Venkataramanan & Shriram 2017, 132-133.)

Muuttujan poistaminen on varmin suorien ja vahvojen epäsuorien tunnisteiden poistamisessa. Esimerkiksi tehtävänimike joka organisaatiossa voi olla yhdellä henkilöllä kerrallaan, kuten rehtori, on syytä poistaa. Toisaalta nimikkeen poisto ei auta, jos poistettu arvo on kohtuullisen helposti pääteltävissä, esimerkiksi toimitusjohtajan tittelin voisi päätellä, jos tulot poikkeavat selkeästi muista, tai vaikka henkilö olisi ainoa, jolta puuttuu sekä titteli ja tulot aineistosta. (Venkataramanan & Shriram 2017, 120.)

Henkilö kannattaa jättää aineistosta pois, jos henkilön anonymisointi ei ole muuten mahdollista, tai yksittäisen tiedon vuoksi muulle aineistolle jouduttaisiin tekemään merkittävästi enemmän suojaustoimia koko datalle. (Venkataramanan & Shriram 2017, 121.)

5.6 Datakeskeiset anonymisointikeinot

Yksi tehokkaimmista työkaluista tiedon suojaamiseen on rajata otanta satunnaisesti, sillä tämän jälkeen ei voida yksittäisen henkilön kohdalla tietää onko tietty henkilö edes datassa mukana (Elliot, Mackey, O´Hara & Tudor 2016, 43). Monet de-anonymisointi menetelmät perustuvat siihen, että saadaan tietoja jostakusta, jonka tiedetään olevan datassa mukana, esimerkiksi jos itse on osa testiaineistoa, tunnistaa itseensä liittyvät tiedot, vaikka ne olisivat suojatutkin.

Aineistoon otetaan vain ne tiedot, jotka ovat testauksen kannalta tärkeitä, eli aineiston sisältö minimoidaan. Jäljelle jäävät suorat tunnisteet korvataan joko generoidulla arvolla tai pseudonymisoimalla. Pseudonymisointi, eli peitearvon käyttö oikean arvon tilalla läpikäydään luvussa 5.7 Pseudonymisointi. Testiaineistossa tietoja halutaan usein mahdollisimman aidonkaltaisina, jolloin poistamisen sijasta voidaan korvata tietoja testiaineistodatalla. Esimerkiksi toimipisteen nimen tilalle voidaan asettaa selkeästi testiaineistokorvikkeet. Ikä voidaan muuttaa raja-arvoissa toimivilla satunnaismuuttujilla, asuinalue ja ammatti voidaan satunnaistaa tunnistamisen estämiseksi.

Mikäli suora korvaaminen ei sovellu tilanteeseen, voidaan lisätä myös kohinaa vähentämään tietojen tarkkuutta, jolloin anonymisoinnin purkamisen kannalta on entistä epävarmempaa tiedon ja siitä tehdyn päätelmän oikeellisuus. Kohinaa voidaan lisätä esimerkiksi muuttamalla aikamuuttujia satunnaisarvolla siten, että datan käyttäjä ei voi tietää onko aineistossa oikea arvo vai ei. Samalla varmistetaan, että jokaisessa ryhmässä on riittävästi tietoa, jotta yksittäinen uniikki tieto ei johda henkilön tunnistukseen. (Venkataramanan & Shriram 2017, 121.)

Testiaineiston suojaamista suunnitellessa huomioidaan kuinka satunnaisia testiaineiston tiedot ovat suhteessa koko aineistoon (Elliot, Mackey, O´Hara & Tudor 2016, 43). Esimerkiksi kokonaisaineistossa voidaan jo etukäteen tietää henkilön sisältyvän mahdollisesti aineistoon, esimerkiksi yrityksen työntekijän voidaan olettaa löytyvän yrityksen henkilöstöhallintojärjestelmästä. Satunnaisesti otetusta aineistosta ei voida olla varmoja onko henkilö mukana aineistossa. Mitä suuremmasta aineistosta testimateriaali otetaan, ja mitä pienempi osuus se on kokonaisaineistosta, sitä vaikeampaa yksittäisen henkilön päättely aineistosta on.

Suojauksen kannalta tärkeintä on tunnistaa avainmuuttujat, joita mahdollinen datan väärinkäyttäjä saa todennäköisimmin haltuun ja joiden perusteella hän voisi luokitella ja rajata dataa. Tiedot aineiston muodostamistavasta ei saa johtaa henkilön tunnistamiseen (Elliot, Mackey, O´Hara & Tudor 2016, 44). Esimerkiksi jos aineiston yhteydessä on tieto mistä yrityksen osastosta tiedot on otettu, mahdollisten henkilöiden määrä rajautuu.

Riippumatta aineiston koosta, aineiston suorat ja epäsuorat tunnisteet käydään läpi harvinaisten tai ainutkertaisten tunnisteiden varalta. Haasteeksi voi muodostua se, että usein halutaan mahdollisimman yksityiskohtaista tietoa jokaiseen tietueeseen, ja juuri harvinaiset tapaukset ovat kiinnostavia testauksen variaatioiden kannalta. (Elliot, Mackey, O´Hara & Tudor 2016, 44.)

Datan turvallisuutta voi parantaa monella tavalla, mutta tärkeintä on, että alkuperäisiä arvoja käytetään vain silloin, kun se on testauksen kannalta tärkeää. Monissa testeissä tärkeämpää on tietorakenteen säilyttäminen, joten arvot voidaan sekoittaa eli satunnaistaa, tietoja voidaan poistaa, tarkkuutta vähentää, karkeistaa ja käyttää vain pieniä otoksia koko aineiston sijasta. (Venkataramanan & Shriram 2017, 21.)

Sotkevia menetelmiä voi käyttää, kun aineistossa on vähän harvinaisia muuttujia. Datan sekoittaminen, eli tiedon siirtäminen tietueelta toiselle toimii hyvin suojaamismenetelmänä, esimerkiksi arkaluontoiset tiedot kuten päivämäärä ja aluetiedot voidaan sekoittaa (Elliot, Mackey, O´Hara & Tudor 2016, 45).

Oikeiden arvojen korvaaminen mallin kautta generoiduilla arvoilla sopii myös testauskäyttöön hyvin. Generointitapa voi vaikuttaa silti suojauksen tasoon: jos alkuperäistä dataa käytetään pitkälle generoinnin pohjana, voi olla helppo päätellä, miten korvaaminen on tehty. (Elliot, Mackey, O´Hara & Tudor 2016, 46).

Pyöristäminen soveltuu datan suojaamiseen, sillä se vähentää tarkkuutta ja vaikeuttaa tiedon kohdentamista tiettyyn henkilöön (Elliot, Mackey, O´Hara & Tudor 2016, 46). Pyöristysluku valitaan arvojoukkoon sopivaksi, mutta esimerkiksi tuntipalkka voidaan pyöristää vaikka lähimpään 5 tai 10.

Taulukossa 2 on koottu luvussa käsitellyt anonymisointikeinot tietojen luokittelun perusteella.

TAULUKKO 2. Testiaineiston anonymisointikeinot tietojen luokittelun mukaan.

Suora tunniste	Epäsuora tunniste	Arkaluontoinen data
Minimointi Poistaminen Korvaaminen Sekoitus Pseudonymisointi *Generaattorit (esim. henkilötunnus, luottokortti,puhelinnumero, sähköposti)	Generointi Sekoitus K-anonymisointi Satunnaistaminen Otoksen rajaus Minimointi	Kohinan lisääminen L-diversiteetti T-Läheisyys Poikkeama käsittely

5.7 Pseudonymisointi

Eniten käytetty anonymisoinnin keino on henkilönimien muuttaminen peitenimiksi eli pseudonyymeiksi. Pseudonyymin tiedon määritys on, että henkilöä ei voi tunnistaa ilman

erillään säilytettäviä lisätietoja. Pseudonyymiksi dataa ei voi kutsua, jos yksittäisen henkilön voi tunnistaa ilman koodattuja suoria tunnisteita kuten henkilötunnusta, esimerkiksi tilanteissa, joissa epäsuorat muuttujat ja harvinaiset ominaisuudet mahdollistavat henkilön tunnistamisen. (Tarhonen 2016, 12.)

Pseudonymisoinnin lähtökohta on korvata tunnisteelliset tiedot joko alkuperäisestä arvoista johdetuilla tai halutulla testiaineistolla niin, ettei henkilö ole enää tunnistettavissa. Tieto alkuperäisistä arvoista ja muutoksen muodostamistavoista pidetään erillään sekä teknisesti, fyysisesti että hallinnollisesti. Käyttöympäristön täytyy olla suojattu, ja käyttöoikeuksien täytyy olla paitsi rajoitetut, myös aktiivisesti valvotut. Teknisillä toimenpiteillä viitataan tiedon kryptaukseen ja tietoturvallisiin tallennusratkaisuihin. (Tarhonen 2016, 10-12.)

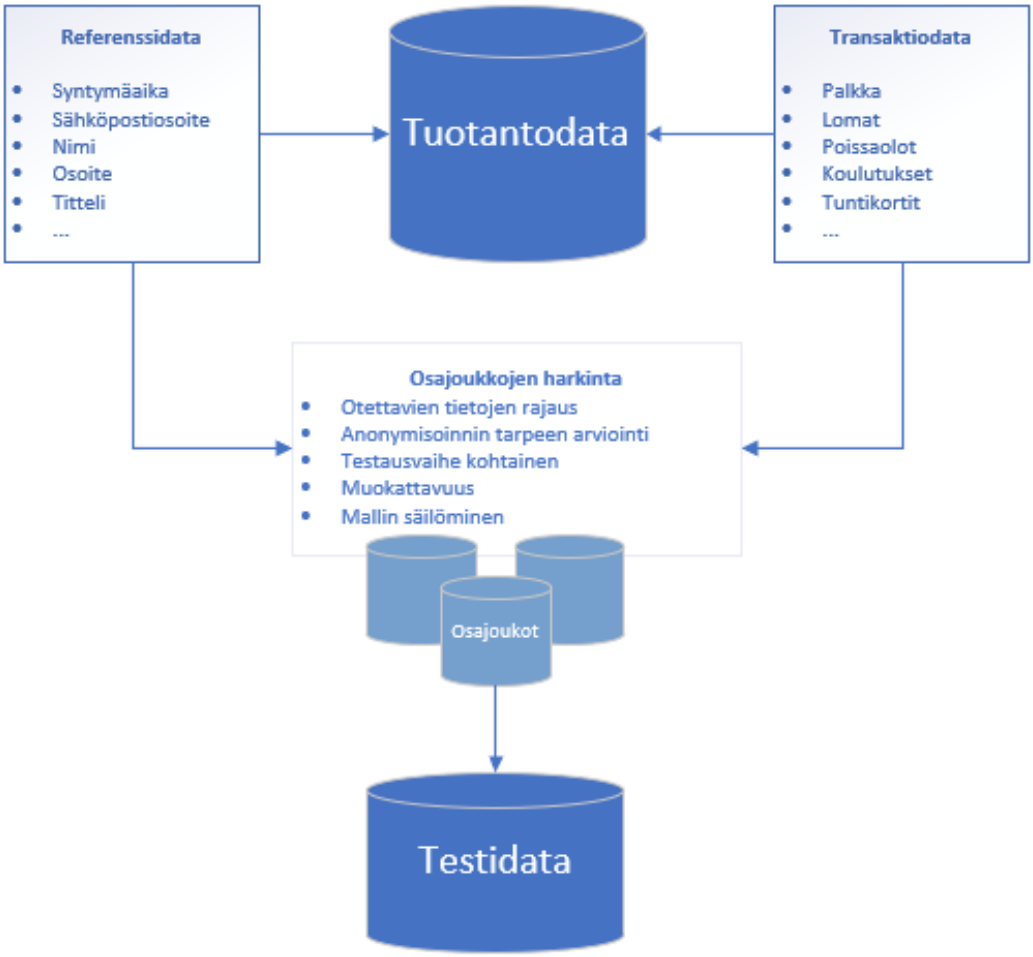
Tunnistetiedot sisältävät koodiavaimen, henkilötiedot ja tietojen muutoksen muodostamistavan. Pseudonyymistä aineistosta tulee anonyymi, kun erillään säilytettävät tunnistetiedot hävitetään. Jos pseudonymisoidun aineiston erillään säilytettäviä henkilötietoja ei voi hävittää, aineistosta voi tehdä anonyymin tuhoamalla koodiavaimen ja muutettujen arvojen muodostamistiedot. Aineisto on hyvä vielä järjestää uudella tavalla, esimerkiksi arvottujen uusien havaintotunnusten mukaan. Aineisto on anonyymi, jos sitä ei voi kohtuullisin keinoin enää yhdistää alkuperäisiin henkilötietoihin. (Tarhonen 2016, 22.)

Pseudonymisoinnissa on ymmärrettävä mistä aineisto tarkalleen muodostuu. Master Data koostuu itse entiteeteistä eli tarkasteltavista asioista, entiteettien välisistä suhteista eli relaatioista ja entiteettien attribuuteista eli niiden ominaisuuksista. Entiteettejä suhteineen ja ominaisuuksineen määrittelevät meta- ja referenssidatat. (Laatikainen 2015.)

Master Data on liiketoiminnan kriittistä tietoa, jota hyödynnetään joka päiväisessä liiketoiminnassa (Laatikainen 2015). Testiaineiston muodostuksessa olemme kiinnostuneita Master Dataan liittyvistä tiedon tasoista kuten transaktiodatasta, eli toimintatiedosta ja itse arvolistoista, eli referenssidatoista.

Kuviossa 11 kuvataan, miten tuotantodatan, eli Master Dataan liittyvät tietokerrokset voidaan johtaa testiaineistoihin sopivaksi. Referenssidata luokittelee muuta dataa, ja on

yleensä organisaation itsensä muodostamaa, eli toistaa organisaation määrittelemiä tietomallia. Transaktiodata kuvaa datan toimintadataa, eli tapahtuma- ja toimintatietoa, kuten palkka- loma ja poissaolotapahtumia. (Laatikainen 2015.)



KUVIO 11 Testiaineiston muodostaminen tuotantoaineistosta

Referenssi- ja tuotantodata muunnetaan testiaineistoksi rajaamalla ensin sopivan aineiston laajuuden. Tämän jälkeen analysoidaan viitatun datan anonymisointitarve, ja tehdään tarvittavat muokkaukset. Otanta tehdään aina testausvaiheeseen erikseen, mutta määritelty malli dokumentoidaan uudelleenkäyttöä ja tarkistamista varten, siksi tehdyt toimenpiteet dokumentoidaan perusteluineen. (Elliot, Mackey, O´Hara & Tudor 2016, 3.)

Dokumentaation päätarkoitus on tukea tiedon elinkaaren hallintaa ja uudelleenkäyttöä. Tiedoista on ilmettävä syy, mihin testiaineistoa käytetään, missä sitä käytetään ja kenen

toimesta. Siitä on tultava ilmi riskienarviointia varten tiedossa olevat ulkopuoliset aineistolähteet ja tietojen julkisuus. Sen lisäksi aineiston luontiaika ja poistamisaika kirjataan. (Elliot, Mackey, O´Hara & Tudor 2016, 3).

Tunnistamisriskiä arvioidaan elinkaaren ajan säännöllisesti, sillä verkossa avoimesti saatavien tietojen määrä lisääntyy koko ajan, kun erilaisia julkisia rekistereitä, verkkosivustoja ja rajapintoja avataan. Uudistuvan ja kehittyvän teknologian muodostamaa riskiä deanonymisoinnille, eli anonymisoinnin purkamiselle, kutsutaan jäännösriskiksi. (Elliot, Mackey, O´Hara & Tudor 2016, 4-5)

Mikäli käsittelyssä on koko aineisto, esimerkiksi kokonainen tietokanta, tai huomattava osa siitä, tarvitaan tunnistamisriskin arviointia varten anonymisointityökaluja. Tunnetuimpia periaatteita ovat k-anonymiteetti ja l-diversiteetti. K-anonymiteetissä estetään henkilön tunnistamista muodostamalla ryhmiä, joilla on vähintään k -määrä epäsuorilta tunnisteiltaan samanlaisia henkilöitä. Esimerkiksi jos aineistossa on 105 vuotias nainen Nokialta, hän ei voi olla ryhmänsä ainoa, jotta k-anonymiteetin vaatimus täyttyisi. K:n arvolle ei ole yksiselitteistä määrää vaan se käsitellään tilannekohtaisesti, mutta lähtökohtaisesti kolme on jo vähimmäismääre. (Venkataramanan & Shriram 2017, 113.)

K-anonymiteetti ei ole riittävä, jos kaikissa k-anonymiteetin omaavissa henkilöissä on jokin arkaluonteinen arvo. Tällöin käytetään tukena l-diversiteettiä, jolla varmistetaan, että arkaluonteisen muuttujan kohdalla on vähintään l-määrä erilaisia arvoja. Esimerkiksi jos l-diversiteetin vaatimus on 2, pitää aineistossa, jossa tutkitaan ulosotossa olevia, olla vähintään l kertaa henkilöitä, jotka eivät ole ulosotossa. (Venkataramanan & Shriram 2017, 113-114.)

Vapaan kirjoituksen muuttujat kuten kommentit, kuvaukset ja lisätiedot, voivat sisältää arkaluontoista tietoa. Tällainen data on syytä poistaa, ellei se testauksen kannalta ole tärkeää. Jos ne päätetään jättää, ne on käytävä läpi tunnistetietojen varalta läpi. Nämä tiedot voi olla myös mielekäs korvata testiaineistolla, jolla saadaan tarvittava vaihtelu tietoihin. (Venkataramanan & Shriram 2017, 117.)

5.8 Anonymisoidun testiaineiston haasteet

Nykypäivänä testiaineistoa tarvitaan nopeasti ja kattavasti, eivätkä ratkaisut testidatan luontiin ole optimaaliset. Testidatan käsittelyyn kuluu paljon aikaa ja datan muodostus voi olla testauksen pullonkaula, jos datan luonti ei ole nopeaa, kattavaa ja riittävän runsasta. Datan generointi edellyttää manuaalista työtä ja ylläpitoa. (Heimola 2017.)

Testidatan anonymisointi on hyvä lähestymistapa yksityisyyden suojaamiseksi. On kuitenkin olemassa tilanteita, joissa testidataa ei voida luoda anonymisoimalla. Tällaisia ovat negatiivinen testaaminen, arkaluontoiset toimialat, ei-toiminnallisten vaatimuksien testaaminen sekä regressiotestaus. Anonymisointi muuttaa testidataa kohti odotettuja arvoja, jolloin tuotantoympäristön puutteet ja poikkeamat datassa peittyvät. (Venkataramanan & Shriram 2017, 115.)

Terveystiedot ja muut arkaluonteiset tiedot ovat alueita, joissa tiedot eivät ole anonymisoitunakaan ole testauskelpoisia, jolloin korkean suojaustarpeen vuoksi on parempi testata manuaalisesti luodulla testidatalla kuin tuotantodatalla. Ei-toiminnallisten vaatimuksien kuten suorituskykyvaatimusten testaaminen täytyy tehdä tuotantokopioidulla datalla tuotantopalvelimella, koska muuten tietoa ei saada asiakkaan ympäristön suorituskyvystä. (Venkataramanan & Shriram 2017, 116.)

Regressiotestausta tarvitaan usein, mutta jos julkaistaan pieniä korjauksia ja muutoksia nopeassa syklissä, anonymisointi voi olla liian raskas toimenpide, koska anonymisoituun testidataan ei voida enää kohdentaa muutoksia, ja anonymisoidun testiaineiston päivitys voi sisältää käsittelylupien anomista, joka hidastaa prosessia. Toisinaan lupaa anonymisointiin ja datan jakoon ei yksinkertaisesti saada. Näissä tilanteissa testidata luodaan synteettisesti analysoiden alkuperäisen datan rakenteita. (Venkataramanan & Shriram 2017, 117-118.)

Aineistojen anonymisoinnin suurin haaste on, että anonymisoinnin kattavuutta tulee arvioida säännöllisesti. Teknologian ja julkisen tiedon kasvaessa mahdollisuudet ja keinot purkaa anonymisointi datasta kasvavat.

6 POHDINTA

Opinnäytteen tarkoituksena oli integroida automaatiotestaus osaksi kehittämistoimintaa ja automatisoida hyväksyntätestausta. Automaatiotestien käyttöönotossa saavutettiin valmius testata, mutta jalkautus on vielä kesken opinnäytteen raportointivaiheessa. Tämä oli kohdeorganisaatiossa tietoinen strateginen valinta, koska resursseja ei pystytty irrottamaan automaatiotestaukseen alkuperäisessä aikataulussa. Toisaalta käsiteltäviä aiheita oli käyttöönotossa paljon ja ne kytkeytyivät muiden projektien aikatauluihin.

Projektissa selkeästi suurimmaksi haasteeksi muodostui aluksi resursointi. Automaatiotestaustiimin sijaan ajateltiin, että yksi henkilö voi rakentaa testiautomaatioon hyvän pohjatestisetin, jota voitaisiin jalkauttaa käyttöön. Ratkaisu olisi voinut onnistua, ellei sama resurssi olisi ollut jakaantuneena moneen muuhun projektiin ja työhön samanaikaisesti, jolloin jäljellä olevalla ajalla testisettien ylläpito ja laajennus osoittautuivat vaikeaksi. Samaan aikaan ymmärrys tuotteen ja testauksen ongelmien syvyydestä laajeni, ja kävi ilmeiseksi, että automaatiotestauksen lisäksi myös monia muita toimintatapoja täytyi muuttaa. Tässä vaiheessa automaatiotestaus kytkeytyi kiinteästi osaksi muuta tuotekehityksen toimintamallin kehittämisprojektia. Lisäksi opittiin, että automaatiotestauksen aloitus vaatii yhdeltä tai useammalta ihmiseltä täydet resurssit aloitusvaiheeseen.

Resursoinnin toinen haaste on se, että kohdeorganisaatio on siirtynyt ketterään tuotekehittämiseen. Viitekehyksenä toimii scrum, mutta koska tiimit ovat vasta siirtyneet uuteen toimintamalliin, tiimeillä ei ollut aikaa automaatiotestaukselle, sillä tiimit kehittivät vielä muidenkin toimintojen osalta toimintatapojaan eivätkä automaatiotestaustaidot olleet osa tiimin osaamisprofiilia. Tiimissä pitää olla testausosaamista määrittelemään epäselvät skenaariot ja testitapaukset, tekemään manuaalista testausta kehittäjien rinnalla ja kirjoittamaan regressiotestit. Testauksen edetessä testausresurssia tarvitaan enemmän, kun asiantuntijataitoja tarvitaan myös integraatio- ja suorituskykytestien kirjoittamiseen. Samaan aikaan regressiotestien määrä kasvaa toimintojen laajetessa, jolloin ylläpidettäviä testejä on yhä enemmän.

Suunnitteluvaiheen päätökset valmistelevat projektin pohjan menestyksen tai epäonnistumiseen. Projektin keskeisiä oppeja oli, että organisaation tulee käyttää riittävästi aikaa tavoitteiden asettamiseen, nykytilanteen ymmärtämiseen ja oikean implementointitiimin

kokoamiseen automaatioprojektin käynnistämiseksi. Mikäli jokin näistä puuttuu, automaatiotestauksen käyttöönotto lähtee epävarmasta asetelmasta. Lisäksi on tärkeää saada johdon aito tuki. Ei riitä, että on tahtotila tehdä, vaan päätöksen teossa projektin tulee olla sellaisella prioriteetilla, että se ei jää kaiken muun työn alle.

Testauksen käyttöönotto tiimiin kannattaa koota organisaation erilaisia asiantuntijoita, jolloin suunnitteluun saadaan arvokasta tietoa eri näkökulmista, mutta myös tuki niin kehittäjiltä kuin testaajiltakin. Helposti testattava koodi on myös helposti ylläpidettävää koodia. Useat automaatiotestauksen kannalta tärkeät toiminnot ovat hyvän koodaustavan mukaisia. Esimerkiksi käyttöliittymätesteissä objektien identifiointi, eli uniikin ID:n asetus, on tärkeää, jotta testi ei hajoa testauskohteesta riippumattomista syistä.

Käyttöönotossa mietittiin työkalujen valintaa testausympäristöön. Mietittiin miten tiimiä voisi tukea muutoksessa ja kouluttaa käyttämään määriteltyjä työkaluja. Tästä seurasi viestintäsuunnitelma, jota muokattiin ymmärtäen, että toimintakulttuuriin oltiin tekemässä muutoksia. Analyysissa ilmeni, että alkuperäinen tavoite automatisoida testausta oli liian väljä määritelmä. Aluksi huomio kiinnitettiin yksikkötestaukseen, sillä yksikkötestit eivät vaadi usein muutoksia, mutta ovat edullinen vaihe saada virheitä kiinni, ja oppimiskäyrä testaukseen on yleensä lyhyt. Yksikkötestaus ei kuitenkaan lähtenyt organisaatiossa jalkautumaan odotusten mukaan, sillä koodipohja on monimutkainen, ja resursseja muutokselle ei ollut.

Tutkimuksessa havaittiin muitakin ongelmia: organisaation muutoksessa tieto ja toiminta oli siiloutunut ja kommunikaatio tapahtui liian myöhään. Tarvittiin nopeita voittoja, jotta automaatiotestaus saisi uskottavuutta. Automatisoitu hyväksyntätestaus käyttöliittymätasolla on nopea keino saada testikattavuutta ja tietoa sovelluksen laadusta. Valittiin käyttäytymisläheinen tuotekehitys, jonka ytimenä on asiakkaan edustajan, tässä tapauksessa tuoteomistajan, kehittäjän ja testaajan kommunikaatio. Yhteinen ymmärrys toiminnoista saavutetaan kirjoittamalla käyttäjätarinat, jotka kirjoitetaan syntaksilla, jonka avulla ne voidaan suoraan ottaa automaatiotestauksen pohjiksi.

Projektin fokus siirtyi hyväksyntätestauksen puolelle, jolloin työkaluksi valittiin aluksi Specflow. Proof of concept -vaiheessa Specflow ei kuitenkaan vastannut teknisiä vaatimuksia projektin aikana, vaikkakin tuki olisi ollut tulossa myöhemmin. Testiympäristöä

vaihdettiin jälleen, tällä kertaa Robot Frameworkkiin. Tukiongelma pystyttiin kiertämään välisovellusten kautta.

Kun samaan aikaan valmisteltiin jatkuvan integraation käyttöönottoa, saavutettiin kokonaisuutena valmius aloittaa testien laajamittainen automatisointi. Automatisoitujen testien ajo vie kauemmin kuin manuaalisten, mutta toistojen kautta ja oikeisiin kohteisiin käytettynä testien automatisointi tuottaa kustannussäästöjen lisäksi varmuutta testauksen laadusta ja regressiosta.

Automatisointi kannattaa aloittaa sieltä missä onnistumisen todennäköisyys on suuri. Alkuvaiheessa automatisoituja testejä kannattaa kehittäjien tehdä osioihin, joissa tila on verrattain vakaa, mutta joihin on pieni muutostarve, jolloin testitapaus on helppo luoda ja rajata, mutta auttaa kehityksessä, jolloin luottamus tuotteeseen kasvaa. Testaajien kannalta testejä kannattaa kohdentaa niihin alueisiin, joissa toistoa on paljon, virheherkkyys suuri ja testaaminen luonteeltaan aikaa vievää, mutta joiden automatisointi ei ole vaikeaa. Suosittelen näitä, koska nopeat tulokset alussa antavat uskoa ja motivaatiota testauksen laajentamiseen muuallekin.

Kaikkea ei ole järkevä automatisoida. Jos testi vaatii paljon resursseja, mutta on manuaalisesti helppo toimenpide, ei manuaalisen osuuden testaamiseen kannata käyttää aikaa, vaan dokumentoida että se ei ole testauksen kohteena. Päätöksen teossa on syytä miettiä minkä tason automaatio testitapauksessa on tarpeen ja miksi ja mitä toiminnallisuudesta ollaan testaamassa. Esimerkiksi on tarpeetonta lisätä käyttöliittymätesteihin sellaisia testejä, jotka tehokkaammin voidaan tehdä yksikkö- tai integraatiotestien kautta, kuten asennuksen toimivuuden testaus.

Testien ylläpidettävyys on syytä pitää mielessä koko ajan, mikä tarkoittaa sitä, että testeistä rakennetaan uudelleenkäytettäviä itsenäisiä yksiköitä, eli atomisia moduuleja ja metodeja. Hyvin tehty testi ei vaadi paljoa ylläpitoa. Automaatiotestien teko kulkee käsi kädessä jatkuvan integraation kanssa, sillä tällöin testejä voidaan ajaa koko ajan.

Edellä esitetyillä suosituksilla on tavoitteena varmistaa, että sijoitetun pääoman tuottoaste on mahdollisimman suuri ja motivaatio tekemiselle korkea. Tuottoasteen laskeminen testaukselle perustuu kulujen vähentämiseen muualta. Kannattavuutta voi maksimoida ra-

kentamalla ratkaisuista monikäyttöisiä, jolloin samoja rakenteita ja työkaluja voidaan uudelleen käyttää myöhemmin myös muiden toimien apuna, kuten testidatan tuottamiseen manuaalista testausta varten tai tuotantoympäristön monitorointiin.

Automaatiotestausprosessin säännöllinen arviointi kannattaa suunnitella jo käyttöönottovaiheessa. Tiimin tulisi arvioida mittarien ja tiimin suorituskyvyn valossa automaatiotestauksen vaikutuksia, ja mitä haasteita toiminnassa on ollut. Miten testausautomaatio roolin toimenkuva on kehittynyt? Mihin testausaika kuluu, onko ylläpitoon käytetty aika kasvamassa? Onko testitulosten läpikäynti ja analysointi helppoa ja nopeaa?

Vastaavaa iterointia ja arviointia käyttämällä työssä havaittiin useita uudistuksia, kuten sen, että normaalista poiketen testiympäristöön testidatan puhdistus oli paras sijoittaa testiajon alkuun eikä lopuksi. Tällä saavutettiin se etu, että testidata on testiajon alkaessa varmasti puhdasta aiemmista testeistä ja jos virheitä tapahtui, virheellinen tilanne oli tulosten analysoijan käsiteltävissä seuraavaan testiajoon asti.

Automaatiotestauksen käyttöönotossa merkittävin muutos prosessiin tapahtui siinä, että testaus haluttiin ottaa mukaan jo aikaisempaa varhaisemmassa vaiheessa. Integraatiotestejä voidaan suunnitella jo määrittelyvaiheessa käyttötarinoiden muodossa ja määrittelystä voidaan johtaa myös yksikkötestauksen vaatimukset jo ennen kuin itse koodia on kirjoitettu riviäkään.

Varsinaisen automaatiotestauksen kannalta suurin haaste on määrittelyvaihe, sillä testausautomaation kirjoittaja tarvitsee vaatimusmäärittelyä ja tietoa eri tasolla kuin tutkiva testaaja, jolla on erinomainen tuotetuntemus. Käyttäytymisläheinen kehittämismalli (BDD) soveltuu näkökulmaksi, sillä käyttäjätarina, joka kuvaa asiakasläheisesti toiminnallisuuden on hyvä vaatimuspohja automaatiotestille. Ketterälle kehittämiselle on tunnusomaista, että vaatimukset elävät ja tarkentuvat iteraatioiden aikana ja käyttäjätarinoita syntyy paljon. Vaatimukset täytyy priorisoida, jotta ensimmäisessä iteraatiossa keskitytään vaatimuksiin, joita ilman ratkaisu ei toimi. Pakolliset ominaisuudet ovat prioriteetiltaan kriittisiä, sillä ilman niitä ominaisuus ei toimi. Nämä ovat myös ensimmäiset automatisoitavat skenaariot, sillä näillä voidaan varmistaa sovelluksen yleiskunto. Seuraava prioriteettitaso sisältää vaatimukset, jotka ovat tärkeitä, mutta eivät toimintaa estäviä. Tyypillisesti myös tämän tason testit automatisoidaan. Ja kolmantena tasona vaatimukset, jotka olisi hyvä olla. Näiden testien automatisointi mietitään tapauskohtaisesti.

Mikäli aloitusvaiheeseen ei osaamista löydy organisaation sisältä, tai resurssin irrottaminen projektiin täydellä painolla on haasteellista, kannattaa pohtia ulkopuolisen osaamisen hakua projektin käynnistämisvaiheeseen.

Projektin aikana muodostui selkeä kokonaiskuva siitä millaisia tarpeita ja testauksen kannalta erityistä huomiota vaativia piirteitä tuoterajapinnoissa on, ja millainen osaamistilanne organisaatiossa on. Osa projektia oli tunnistaa ohjeistus, koulutus ja perehdytys tarpeita ja laatia materiaalit, joiden avulla osaamista voidaan jakaa. Projektissa tärkein saavutus oli löytää ja pilotoida ratkaisut organisaation tarpeisiin keskittymättä ainoastaan teknisiin vaatimuksiin, vaan miettien kokonaiskulttuurimuutosta. Käyttäytymisläheisellä kehittämistavalla ja käyttäjätarinalähtöisillä skenaariokuvauksilla voidaan avata kommunikaatiota organisaation sisällä. Ratkaisumallissa automaatiotestauksen tulokset ja kattavuus ovat kenen tahansa luettavissa, mikä tuo läpinäkyvyyttä ja mitattavuutta koko prosessiin. Opinnäytteen suurimpia saavutuksia oli muodostaa visio siitä mihin käyttöönotolla pyritään ja valita työkaluista sellaiset, jotka sopivat juuri tämän organisaation käyttöön ja tarpeisiin. Toisaalta huomiota kiinnitettiin paljon myös testauskulttuurin muodostumiseen vaikuttaviin tekijöihin ja suhtautuminen testauksen automatisointiin on organisaatiossa positiivinen.

Automaatiotestauksen käyttöönoton jälkeen jatkokehityskohteita on useita. Ensimmäisiä toimia on laajentaa pilotointi ja käyttöönottovaiheiden testejä ja tätä varten on hyvä asettaa tavoitteita. Yksi suunniteltu rajapyykki on ydinprosessin hyväksyntätestauksen automatisoinnin kattavuus tasolle, jolla se voidaan ottaa yhdeksi version kypsyyden mittariksi. Jatkuvan integroinnin haasteet korostavat automaatiotestauksen tärkeyttä, joten tämän opinnäytetyön pohjana ollut käynnistysprojekti on tärkeä askel kohti tulevaisuutta.

LÄHTEET

Banerjee, K. 2018. Test Automation – A Recipe for Continuous Delivery Success. Luettu 6.5.2018. https://dzone.com/articles/test-automation-a-recipe-for-continuous-delivery-s

Elliot, M., Mackey, E., O´Hara. K. & Tudor, C. 2016. The Anonymisation Decision-Making Framework. Manchester: UKAN.

Euroopan parlamentin ja neuvoston asetus (EU) 2016/679, EU:n virallinen lehti, 27.4.2016. http://eur-lex.europa.eu/legal-content/FI/TXT/PDF/?uri=CELEX:32016R0679&from=FI

Fewster, M & Graham, D. 1999. Software test automation, effective use of test execution tools. New York: Addison-Wesley.

Ghahrai, A. 2018. Problems with Test Automation and Modern QA. Luettu 30.11.2018. https://www.testingexcellence.com/problems-test-automation-modern-qa/

Heimola, A. Senior Consultant. 2017. GDPR and Test Data Challenge. Koulutus. Test Automation Clinic. 4.5.2017. Qentinel Oy. Helsinki.

Heimola, A. Senior Consultant. 2017. Testiautomaatio ROI – Return on Investment. Koulutus. DevOps perehdys 7.4.2017. Accountor HR Solutions Oy. Tampere.

Kasurinen, J. P. 2013. Ohjelmistotestauksen käsikirja. Jyväskylä: Docendo.

Laatikainen, T. 2015. Master data on monisyistä, monista syistä. Luettu 1.10.2018. https://www.arihovi.com/master-data-blogi/

Loveland, S., Miller, G., Prewitt, R. & Shannon, M. 2004. Software testing techniques: finding the defects that matter. Weston: Charels River Media.

McPeak, A. 2018. The Basics of BDD in Testing. Luettu 24.3.2018. https://dzone.com/articles/the-basics-of-bdd-in-testing

Myers, G. Sandler, C. & Badgett, T. 2012. The Art of Software Testing. 3. painos. Hoboken, New Jersey: John Wiley & Sons, Inc.

Nabil, M. 2017. The Automation Testing and Agile. Luettu 1.7.2018. https://medium.com/@Moatazeldebsy/the-automation-testing-and-agile-7a8a8c983ed0

Philips, J., Brantley,W. & Philips, P. 2012. Project Management ROI : A Step-by-Step Guide for Measuring the Impact and ROI for Projects. Hoboken, N.J.: Wiley.

Ponteva, K. 2010.Onnistu muutoksessa. Juva: WS Bookwell Oy.

Rowe, S. 2014. A Brief History of Test. Luettu 24.3.2018. https://blogs.msdn.microsoft.com/steverowe/2014/06/04/a-brief-history-of-test/

Smart, F. 2015. BDD in Action. Shelter Island:Manning.

Tarhonen, L. 2016. Pseudonymisation of Personal Data According to the General Data Protection Regulation. Viestintäoikeuden vuosikirja 2016.

Simbhoedatpanday, K. 2015. Future of Testing and Automation: The role of the tester in 2020. Luettu 21.2.2018. http://blog.xebia.com/future-of-testing-and-automation-the-role-of-the-tester-in-2020/

Venkataramanan, N. & Shriram, A. 2017. Data Privacy: Principles and Practice.

LIITTEET

Liite 1. Yksikkötestaajan muistilista

Yksikkötestaus kohde	Esimerkkejä testitapauksesta
Syötettyjen arvojen tyyppi	Virheellinen tyyppi, esimerkiksi lukutyyppiin tulee kirjaimia. Alustamattoman muuttujan käyttäminen. Arvo puuttuu, null, tyhjä
Syötettyjen arvojen rajat	Liian pienet ja liian suuret arvot Raja-arvo Odotetaan tietynlukujoukon jäsentä, mutta tulee arvojoukkoon kuulumaton arvo
Käyttäjän syöttämät arvot	Kielikulttuuri Moniasiakasympäristöt Ryhmät Käyttöoikeusrajaukset Useita samoja kutsuja
Toistorakenteiden rajat	Ylivuoto, esim. Datasetin taulun rivioletus
Valintarakenteet	Voiko kaikki rakenteen eri vaihtoehdot toteutua?
Komponenttien rajat	Mitä moduuli tekee, jos viestien ja kutsujen lähettäminen ei onnistu? Entä jos vastaanottaminen ei onnistu? Mitä jos nousee virheitä?
Käyttäjän valitsema järjestys tapahtumille	Mitä jos käyttäjä ei noudata oletus suoritusjärjestystä?
Näkyvyysrajat	Pystytäänkö kutsuja kutsumaan kaikkialta mistä niitä pitää pystyä kutsumaan? Pystyykö kutsuja tekemään ohi normaali rakenteen? (tietoturva!)
Syntaksivirheet	Onko muuttujien nimet kirjoitettu oikein (javascript) Onko ohjelmointikäytäntöjä noudatettu?
Erikoismerkkien käsittely	Miten komponentti selviää esim. End-of-file merkistä,
Muistinkäsittely	Varaus Viittaus ryhjään Muistin vapautus

Liite 2. Datan anonymisoinnin muistilista

Tunnistetieto	Suora tunniste	Vahva epäsuora tunniste	Epäsuora tunniste	Anonymisointi-tekniikka
Henkilötunnus	x			Poista
Koko nimi	x			Poista/Muuta
Sähköposti-osoite	x	x		Poista
Puhelinnumero		x		Poista
Postinumero			x	Poista/Luokit-tele
Kaupunginosa			x	Luokittele
Asuinkunta			x	Luokittele
Maakunta			x	(Luokittele)
Suuralue			x	
Puhetallenne	x			Poista
Videotallenne henkilö(i)stä	x			Poista
Valokuva hen-kilöstä	x			Poista
Syntymäaika		x		Luokittele

Ikä			x	Luokittele
Sukupuoli			x	
Siviilisääty			x	
Perheen koostumus			x	(Luokittele)
Ammatti		(x)	x	Luokittele
Toimiala			x	
Työmarkkina-asema			x	
Koulutus			x	Luokittele
Koulutusala			x	
Äidinkieli			x	Luokittele
Kansallisuus			x	(Luokittele)
Työpaikka		(x)	x	Luokittele
Auton rekisteri		x		Poista
Tutkittavan julkaisun/teoksen nimike		x		Luokittele
Verkkosivun osoite		(x)	x	Poista
Henkilönumero		x		Poista
Vakuutusnumero		x		Poista
Tilinumero		x		Poista
Tietokoneen IP-osoite		x		Poista
Terveyttä koskevat tiedot *		(x)	x	Luokittele/Poista
Etninen alkuperä *		(x)	x	Luokittele/Poista
Rikos tai saatu rangaistus *			x	Luokittele/Poista

Ammattiliiton jäsenyys *			x	Luokit-tele/Poista
Poliittinen tai uskonnollinen vakaumus *			x	Poista
Muu luotta-mustoimi tai jäsenyys		(x)	x	Poista
Sosiaalihuol-lon tarve *			x	Poista
Sosiaalihuol-lon tukitoimet ja etuudet *			x	Poista
Seksuaalinen suuntautumi-nen *			x	Poista

Liite 3. Anonymisoinnin läpikäynnin muistilista

Anonymisoinnin menetelmän valintaa ja toteutuksen onnistumista voi kuitenkin arvioida tehokkaasti seuraavien EU:n tietoryhmän lausunnosta (WP 29 lausunto 05/2014) poimittujen kysymysten avulla. Jos vastaus kahteen ensimmäiseen on kielteinen ja viimeisessä todennäköisyys päättelyyn hyvin pieni, aineiston anonymiteetti on hyvällä mallilla.

Onko tunnistetaulukko käyty aineiston suhteen läpi?

Havainnon erottaminen joukosta: Voiko anonymisoinnin jälkeen henkilö olla edelleen tunnistettavissa aineistosta?

Yhdistettävyys: Voiko vastaajien tiedot yhdistää toiseen aineistoon tai ulkopuoliseen tietoon, ja mahdollisesti sitä kautta tehdä henkilöitä tunnistettaviksi?

Päättely: Millä todennäköisyydellä anonymisoitujen muuttujien arvoista voi päätellä muuttujan alkuperäisen arvon?