Osaamista
ja oivallusta
tulevaisuuden
tekemiseen

Sami Penttinen

Testaus Ohjelmistokehityksessä

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tieto- ja viestintätekniikka

Insinöörityö

08.11.2019

Tekijä Otsikko Sivumäärä Päivämäärä	Sami Penttinen Testaus ohjelmistoprojektissa 24 sivua 08.11.2019
Tutkinto	Insinööri (AMK)
Tutkinto-ohjelma	Tieto- ja viestintätekniikka
Ammatillinen pääaine	ohjelmistotuotanto
Ohjaajat	Janne Salonen Jussi Alhorinne

Opinnäytetyössä käsitellään testausta kokonaisuutena ohjelmistokehitysprojektissa. Tavoitteena oli luoda aiheesta täysin tietämättömälle lukijalle selkeä kokonaiskuva laadunparannusprosessin kulusta ja sen eri vaiheista. Työssä käsitellään myös hyväksi todettuja toimintatapoja ja mahdollisia kompastuskiviä kokeneempaa lukijaa varten.

Opinnäytetyö pohjautuu hyvin pitkälti omaan projektikokemukseen ja projektissa käytyihin prosessin kehityskeskusteluihin, joihin osallistui alan ammattilaisia kehitys-, testaus- ja tuotehallinnan puolelta. Työn merkittävämpiä osia oli tiedon kerääminen, ongelmien juurisyiden selvittäminen ja ratkaisujen löytäminen.

Työn suurimmaksi ongelmaksi nousi tiedonkeräämisen hitaus. Johtuen projektin suuresta koosta, muutosten läpivientiin ja tulosten keräämiseen meni huomattavan paljon aikaa. Tämän takia tuloksia muutosten vaikutuksista joutui usein odottamaan viikkoja, jonka jälkeen saatettiin vielä tehdä uusia parannuksia prosessiin.

Testausprosessin parantamisen tuloksena saatiin tehostettua ja selkeytettyä jokapäiväistä projektityöskentelyä. Suurimmat vaikutukset tulivat selkeistä yhteisistä toimintatavoista ja kommunikaation vaivattomuudesta. Työn vaivaton sujuminen vaikutti myös positiivisesti projektin ilmapiiriin.

Avainsanat	ohjelmistotuotanto, manuaalinen- ja automaatiotestaus, testausprosessi

Author Title Number of Pages Date	Sami Penttinen Testing in software project 24 pages 08.11.2019
Degree	Bachelor of Engineering
Degree Programme	Information and communication technologies
Professional Major	Software engineering
Instructors	Janne Salonen Jussi Alhorinne

This thesis deals with testing as a whole in a software project. The goal was to give the reader who is not familiar with the subject a fundamental foundation of a software testing as a process. The thesis also covers proven practices and most common stumbling blocks for more experienced reader.

The thesis is based on experiences and discussions of improving the working practices. These meetings involved professionals from development, testing and product management departments. The most important part of the work was to identify the root causes of problems and innovate working solutions to correct them.

The most noticeable problem was the time it took to get results. Due to the large size of the project it took considerable long time to get to the point of collecting results. Every time improvements were made it often took couple of weeks to notice the effects of the change only to initiate another iteration of the solution.

The Improvements in the testing process resulted in more efficient and pleasant day-to-day working culture in the project. The greatest impact came from more efficient communication and improved teamwork. Proficiently running project also had a positive impact on the overall atmosphere.

Keywords	software engineering, manual testing, test automation, testing processes

Sisällys

1 Johdanto

Ymmärtääksemme mistä kaikki alkaa, meidän tulee tarkastella syitä, jotka johtavat virheisiin ohjelmistokehityksessä. Virheiden tapahtuminen on ihmiselle yleinen asia, joka korostuu etenkin silloin, kun kyseessä on tiukat aikataulut, monimutkaiset järjestelmät ja vaihtuvat teknologia. Nämä ovat yleisimmät syyt, jotka johtavat virheisiin järjestelmien suunnittelussa sekä sen toteutuksessa ja lopulta aiheuttaa ohjelmistoissa tapahtuvia virhetilanteita. Jos järjestelmä suunnitellaan dokumentin pohjalta, jossa on jo virhe, niin on hyvin todennäköistä, että sama virhe siirtyy myös toteutusvaiheessa ohjelmistoon. On myös muitakin syitä kuin ihmisen tekemät virheet sille, miksi järjestelmät ajautuvat virhetilanteisiin. Ympäristön olosuhteet vaikuttavat myös järjestelmien toimintaan. Tällaisia tekijöitä ovat muun muassa lämpötilat, magnetismi tai sähkökenttien aiheuttama häirintä laitteen toiminnalle. Esimerkkinä laitteesta, johon kohdistuu hyvin voimakkaita ja erilaista ympäristövaikutuksia, voidaan pitää avaruudessa olevaa satelliittia. Tosin suurin osa meidän jokapäiväisessä käytössä olevista laitteista on hyvin neutraalissa ympäristössä, joten keskitymme itse koodin tuotantoprosessin parantamiseen.

Jos haluamme parempaa laatua ja minimoida virhetilanteet, meidän tulee välttää virheiden päätymistä järjestelmiin tai löytää ne ajoissa korjataksemme ne. Testaus on erittäin hyvä lähestymistapa laadun varmistamiseksi. Testaus itsessään ei lisää järjestelmän laatua, vaan se antaa testattavan järjestelmän toiminnasta tietoa, jonka perusteella voidaan tehdä arvioita ja päätöksiä seuraavista toimenpiteistä. Järjestelmä vodiaan siirtää tuotantoon käytettäväksi vasta, kun testauksesta on saatu varmuus laatukiriteerien täyttymisestä.

Järjestelmästä riippuen sen virheetön ja vakaa toiminta voi olla erittäin tärkeässä tai merkityksettömässä asemassa. Järjestelmässä esiintyvä vikatilanne voi aiheuttaa suuriakin menetyksiä. Tällaisia ääripään menetyksiä voivat olla, ihmishenget, taloudelliset tappiot, maineeseen ja ympäristöön kohdistuvat vahingot. Yhtenä esimerkkinä tällaisesta voisi mainita lentokoneiden laitteiston, jonka viallinen toiminta voi johtaa lentokoneonnettomuuteen, josta voi seurata lentoyhtiölle menetyksiä kaikilla yllämainituilla osa-alueilla. Toisen ääripään esimerkkinä voi mainita tietokonepelit, joissa ohjelmiston virheen merkitys ei aiheuta vaaratilannetta. Näin ollen ohjelmistojen eri kriittisyysasteesta riippuen on tärkeää osata asettaa tilannetta vastaavat laatu- ja testausvaatimukset.

2 Testauksen tavoitteet

2.1 Testauksen tavoitteet

Jokaisessa ohjelmistoprojektissa riskin mahdollisuus on läsnä. Järjestelmä ei toimi tai se ei valmistu ajallaan. Nämä asiat korostuvat monimutkaisuuden myötä. Oletuksena odotamme, että näitä monimutkaisia järjestelmiä, kuten lentoliikenteen järjestelmiä on testattu enemmän kuin esimerkiksi videopelien, koska riski on suurempi. Näin ollen testauksen astetta tulee säätää riskin mukaan.

Laatu on erittäin vaikeasti määriteltävä asia, koska siihen liittyy niin monta eri asiaa ja näkökulmaa. Hyvänä laadun alkutavoitteena jokaiselle ohjelmistolle on täyttää käyttäjän perustarpeet. Vaikka jokaisella loppukäyttäjällä on oma henkilökohtainen näkemyksensä, miten tuotteen tulisi toimia ja miltä sen pitäisi näyttää, on tärkeää heti projektin alussa määrittää, mihin suuntaan projektia lähdetään viemään. Kokonaisuuden selkeyttämiseksi laadun määritykset voidaan jakaa kahteen eri osa-alueeseen: toiminnallisiin ja ei-toiminnallisiin ominaisuuksiin. Toiminnalliset vaatimukset määrittelevät, miten ohjelmiston ominaisuuksien tulisi käyttäytyä eri tilanteissa ja mitä kaikkea niillä voi tehdä. Ei-toiminnalliset vaatimukset vuorostaan määrittävät tietoturvaan, skaalautuvuuteen ja ohjelmiston saatavuuteen liittyvät asiat.

Vaikka haluaisimme suorittaa kaiken kattavan testauksen ohjelmistollemme, sen täydellinen toteuttaminen ei ole pelkästään mahdotonta, mutta myös liian hidasta ja liikaa resursseja kuluttavaa. Testauksen perimmäinen tarkoitus on kuitenkin tehdä säätöä löytämällä virheet mahdollisimman aikaisessa vaiheessa, jotta virheiden lopullinen vaikutus saadaan minimoitua. Myös virheen korjaaminen on sitä halvempaa, mitä aikaisemmassa vaiheessa kehitystä se löydetään. Näin ollen testauksen tärkein asia on priorisointi, jotta saavutetaan tarvittava hyöty rajallisesta määrästä. Testaus tulisi suorittaa aina tärkeimmistä kohteista alaspäin. Tällöin, jos jostain syystä kuten ajallisesta syystä, osa testaukseen varatusta ajasta jää pois, niin voidaan silti olla varmoja, että kaikki tärkeimmät testit on suoritettu ja isoimmat riskit katettu. Tärkeimpiä testauksen kohteita ovat ne ominaisuudet, jotka vaikuttavat kriittisesti asiakkaan toimintaan. Myös testauksen rajaus on tärkeä osa testausprosessia, että tiedetään, milloin on testattu tarpeeksi eikä testaus veny loputtomiin, mikä syö resursseja. Yleensä tätä varten tehdään erikseen määritetty kriteerit, jotka määrittävät, mikä on valmiin määritelmä. Näin ollen tiedetään lopettaa oikeaan

aikaan. Prioriteetit ja valmiin määritelmä muodostavat keskeisen osan testauksen suunnitelmasta. (2, s. 18, 19.)

2.2 Mitä testaus on ja mitä se tekee

Testaus on järjestelmällistä komponentin tai ohjelmiston tutkimista, jonka tarkoitus on löytää ja dokumentoida virheitä ja puutteita ohjelmiston toiminnassa. Testaus usein liitetään myös virheiden korjaukseen, mutta näin ei ole. Testauksen tarkoitus on tutkia ja antaa informaatiota päätösten tekoa varten. Päätös virheiden korjaamisesta tehdään sen prioriteetin perusteella tai voidaan jättää jopa korjaamatta, jos se ei vaikuta kriittisesti ohjelman toimintaan. Varsinainen virheiden korjaaminen kuuluu osana kehitystyötä, joka on erillinen osa-alue ohjelmistokehitystä kuin testaus. Testaustyö ei kuitenkaan lopu virheen osalta siihen, että se on löydetty ja dokumentoitu, vaan testaukseen kuuluu varmistaa, että korjatut virheet on korjattu oikein ja ettei korjaus ole aiheuttanut uusia virheitä. Ensimmäinen mielikuva testauksesta on yleensä koodin suorittamiseen tai ohjelmiston koe käyttämiseen liittyen eli dynaamiseen testaukseen. Dynaamisiin testaus on hyvin keskeinen muttei ainoa testausmenetelmä. Testausta voi myös tehdä ilman tai jopa ennen kuin yhtään koodiriviä on kirjoitettu. Tällaisia testausmenetelmiä kutsutaan staattisiksi menetelmiksi. Yleisin staattinen testausmuoto on katselmointi. Katselmoinnissa tuotettu dokumentti käydään läpi jonkun toisen kuin sen tuottaneen henkilön toimesta, jolloin saadaan toisen henkilön näkökulma asiaan ja samalla esiin voi nousta yksinkertaisia virheitä. Katselmoinnin tarkoitus on löytää virheitä mahdollisimman aikaisessa vaiheessa, jolloin ne on helpoin korjata. Katselmointia voi suorittaa juuri tuotettuun koodiin tai mihin tahansa projektin dokumenttiin. Halvimmat virheet korjata on ne, jotka löydetään jo dokumentointi vaiheessa ennen kuin ne kerkeävät toteutua osaksi ohjelmistoa. (1, s. 20.)

Testaus on prosessi niin kuin mikä tahansa kehitystyö. Ennen kuin voidaan aloittaa testien suorittaminen, on tehtävä paljon valmisteluja. Ensimmäinen tehtävä on määrittää, mitä testauksella halutaan hakea, jotta voidaan asettaa selkeät tavoitteet testauksen suorittamiselle ja testauksen suunnitteleminen voi alkaa. Testisuunnitelma tekeminen on tärkeä osa testausta, sen avulla voidaan helposti käydä läpi, mitkä vaiheet tullaan testaamaan ja missä vaiheessa. Testisuunnitelma antaa myös luotettavuutta testaukselle.

Siitä nähdään, ettei mitään kriittisiä vaiheita ole jäänyt välistä ja se toimii myös helposti ymmärrettävänä dokumenttina asiakkaalle, josta asiakas voi tehdä päätöksen onko testausta tehty tarpeeksi kattavasti. Kun kaikki valmistelutyöt on tehty voi varisnaisten testien kehitys ja suorittaminen alkaa. Testien suorittamisen jälkeen on vielä jäljellä testauksen tärkein vaihe eli tulosten raportointi ja dokumentointi, joka on tärkein testauksesta saatu hyöty. Tämän hyödyn pohjalta saadaan tietoa tuotteen laadusta ja voidaan tehdä päätöksiä sen parantamiseksi ja virheiden korjaamiseksi.

3 Dynaamisen testauksen osa-alueet

3.1 Tietoturvatestaus

Laadukas ohjelmisto ei ole vain toiminnallisuudeltaan laadukas vaan se on myös turvallinen käyttää. Tietoturvatestauksen tavoitteena on tehdä ohjelman käyttö mahdollisimman turvalliseksi ja estää mahdolliset väärinkäyttötapaukset. Tietoturvatestaus on siinä mielessä hankalaa, että tietoturva-aukkojen huomaaminen on vaikeaa. Ohjelmisto voi toimia täysin oikein, vaikka siinä olisi tietoruvapuutteita, jotka eivät näy ohjelmiston käyttäjälle. Tietoturvapuutteet, jotka tulevat esille vasta tuotteen käyttöönoton jälkeen, voivat olla hyvinkin suurivaikutteisia.

Tietoturvaan liittyvät vaatimukset määritellään heti projektin suunnitteluvaiheessa, ja näin ollen voi sen testauksenkin aloittaa hyvin aikaisessa vaiheessa. Tietoturva ja sovellusvaatimusten perusteella voidaan luoda arkkitehtuuri mallisovelluksesta, jonka pohjalta voidaan aloittaa tietoturvatestaus jo ennen sovelluksen kehittämistä. Tietoturvatestaus voidaan aloittaa tekemällä uhkamallinnus. Uhkamallinuksessa sovellusta voidaan tarkastella palvelun suojaajan sekä hyökkääjän näkökulmasta. Kun sovellusarkkitehtuuri on tiedossa, niin voidaan hyvinkin tarkkaan miettiä, minkä kaltaisille hyökkäyksille sovellus voi olla alttiina. Näistä hyökkäysanalyyseistä saadaan hyviä testitapauksia testaukselle ja samalla voidaan pohtia, kuinka näitä hyökkäyksiä tullaan estämään. Tietoturvan kehitys on koko prosessin kestävä tapahtuma ja on tärkeää aloittaa se projektin alkuvaiheessa, koska sen tekeminen jälkeenpäin voi olla hyvin kallista tai jopa mahdotonta. Tietoturvatestauksessa käytetään yleensä paljon analyysityökaluja, jotka tarkastelevat ohjelmistoa mahdollisten tietoturva-aukkojen kannalta. Yleensä näissä työkaluissa on kartoitettu tunnetuimmat haavoittuvuudet, ja ne skannaavat tarkastettavan ohjelman niiden varalta. Tietoturva on myös koko ajan muuttuvaa, koska ohjelmistojen toiminnassa

käytettäviin kirjastoihin tulee uusia muutoksia ja versioita. Yksi tietoturvatyökalujen tärkein tehtävä onkin varmistaa, että käytössä on aina parhaaksi todetut ohjelmistoversiot eikä vanhentuneita komponentteja käytetä.

3.2 Yksikkötestaus

Yksikkötestaus on ensimmäinen dynaamisen testauksen taso monella tapaa. Se tapahtuu ensimmäisenä ja matalimmalla tasolla. Yksikkötestauksessa testaan juuri tuotettujen koodin osien, kuten funktioiden ja luokkien toimintaa. Tarkoitus on varmistaa pienimmän mahdollisen koodikokonaisuuden toiminta täysin eristettynä kaikesta muusta toiminnallisuudesta, jotta saadaan täysi varmuus osan täydestä toimivuudesta ilman ulkoisia vaikutuksia ennen kuin se liitetään isompaan kokonaisuuteen. Yksikkötestauksen tulisi aina keskittyä vain yhteen testattavan komponenttiin kerrallaan.

Koska yksikkötestaus on matalimmalla tasolla suoritettavaa testausta, tulisi se näin ollen suorittaa heti kehityksen yhteydessä, ettei viallinen koodi pääse vaikuttamaan jatkokehitykseen. Yksikkötestauksen tärkeimpiin ominaisuuksiin kuuluu sen nopeus ja kustannustehokkuus. Kun virheet tulevat ilmi heti kehitys työn yhteydessä, on ne nopea korjata eivätkä virheet pääse vaikuttamaan muun projektin työntekoon. Yksikkötestaus tapahtuu usein saman kehittäjän toimesta, joka on myös kirjoittanut kyseisen koodin osan ja näin ollen luokitellaan niin sanotuksi lasilaatikkotestaukseksi, koska kehittäjä näkee ja tietää kyseisen osan sisältävän tapahtuvat toiminnot. Yksittäisen koodin osan testaamiseen voidaan hyödyntää testaukseen tarkoitettuja laajennusominaisuuksia, joilla voidaan nauhoittaa testaukseen käytettyjä arvoja ja saatuja tuloksia sekä laskea suoritukseen kulunutta aikaa. Näiden lisäominaisuuksien avulla on kehittäjän helppo luoda testejä, jotka syöttävät testattavaan osaan syötteitä ja tarkastaa, että komponentti palauttaa halutun arvon. Yleisimmät virheet, joita ilmenee yksikköteistauksessa toiminnallisuuden osalta, ovat virheellinen arvojen käsittely ja virheellinen tai jopa kokonaan puuttuva ohjelmointipolku, jotka ovat yleisiä konditionaalityyppisten lauseiden yhteydessä. Yleensä tällaiset virheet tulevat ilmi, kun testauksessa siirrytään tavallisesta tapauksesta niin sanottuun negatiiviseen tapaukseen, eli mitä ei saisi tapahtua tiettyjen syötteiden kohdalla. Tämä tapahtuu usein etenkin ohjelmointipolkujen kohdalla, kun kehitys hetkellä ei ole aina helppoa huomata kaikkia mahdollisia vaihtoehtoja, joita voi tapauksesta riippuen olla lukuisia. (1, s. 97.)

3.3 Integraatiotestaus

Yksikkötestauksen jälkeen, kun yksittäisen osan toiminnallisuus eristetyssä ympäristössä on varmistettu, on sen aika integroitua eli liittyä osaksi isompaa kokonaisuutta. Nyt kun monta yksittäistä osaa on yhdistetty isommaksi kokonaisuudeksi, on varmistettava, että niiden yhteistoiminta ja kommunikointi pelaavat suunnitellulla tavalla. Vaikka kyseinen osa on juuri yksikkötestattu ja sen sisäinen toiminta varmistettu on silti syytä testata sen toiminta osana kokonaisuutta, missä datan vastaanotto ja lähetys seuraavalle yksikölle on keskeisessä osassa. (2, s. 46.)

Ideaalisesti integraatiotestaus tapahtuu aina askel kerrallaan jokaisen osan liittämisen kohdalla kohti monen komponentin sisältävää integraatiotestikokonaisuutta. Näin saadaan heti palautetta viallisen komponentin kohdalla eikä aikaa kulu läheskään yhtä paljon ongelman paikantamiseen. Integraatiotestaus tapahtuu hyvin pitkälti samaan malliin kuin yksikkötestaus. Yksikkötestauksen apuohjelma tulisi valita siten, että samaa työkalua ja testejä voidaan hyödyntää integraatiotestauksessa ajan ja vaivan säästämiseksi. Integraatiotestauksen ero yksikkötestaukseen on syötteen tarkastamisen kohta. Kun yksikkötestauksessa syöte tarkastetaan heti samasta moduulista ulos tullessa, niin integraatiotestauksessa sen annetaan kulkea vielä yhden tai useamman moduulin läpi ennen kuin sen oikeellisuus tarkistetaan. Näin saadaan varmuus, että moduulit osaavat kommunikoida oikein keskenään. (2, s.49, 50.)

Yleisemmät virheet, joita ilmenee integraatiotestauksessa, ovat komponenttien kommunikaatio-ongelmat, tai niiden lähettämä data on vääränmuotoista vastaanottavalle komponentille. Näin ollen ne eivät pysty sitä käsittelemään. Vääränmuotoinen tai puuttuva data kriittisessä kohdassa koodia voi aiheuttaa koko järjestelmän kaatumisen ja näin aiheuttaa suurtakin haittaa käyttäjälle. Myös aika, joka kuluu ohjelmistotoimintojen välillä, voi olla iso vaikuttava tekijä ohjelmisto kokonaisuuksissa. Jos ohjelma on aika kriittinen ja osien kommunikointi tapahtuu väärään aikaan tai liian hitaasti, saattaa se aiheuttaa ongelmatilanteita. Tämän takia integraatiotestaus on erittäin tärkeä osa testauskokonaisuutta. Kaikki nämä ongelmat olisivat jääneet täysin testaamatta pelkällä yksikkötestauksella, vaikka se olisi suoritettu kuinka hyvin tahansa. Tässä kohtaan joku saattaa ajatella, miksi yksikkötestaus on tarpeellista, jos tämän kaiken pystyy suorittamaan integraatiotestauksellakin. Suurin hyöty on yksikkötestauksen nopeampi reagointi ja tarkempi vian paikannus verrattuna integraatiotestaukseen ja näin säästää työkustannuksissa.

Integraatiotestauksessa on myös erittäin tärkeää huomioida järjestys, jossa komponentteja testataan, että testaus sujuisi nopeasti ja vaivatta testaus kustannusten optimoimiseksi. Tämä on yleensä testauspäällikön tehtävä päättää, missä järjestyksessä osat liitetään parhaan strategian luomiseksi kehitykselle. Vaikeudeksi tulee se, että osien kehityksen välillä voi mennä viikkoja tai jopa kuukausia ennen kuin uusi osa liitetään kokonaisuuteen, mutta testaus tiimillä pitäisi olla koko ajan työtä, ettei turhia kustannuksia kerry. Pääsääntönä on, että pyritään kehittämään keskeiset osat ensimmäisenä, jota kaikki muut osat käyttävät, ja sitten rakennetaan tämän keskeisen komponentin ympärille kaikki muu toiminto. Näin vältetään isoimmat ongelmakohdat, koska on huomattavasti helpompi vaihtaa pieni ohjelmistoon liitetty osio, joka ei ole kriittinen muun toiminnallisuuden kannalta kuin ohjelman keskeinen osa, jonka ympärille kaikki perustuu. Testauksen kannalta on mahdollista myös jäljitellä järjestelmään liitettävien komponenttien toiminnallisuutta. Näin voidaan ennakkoon imitoida järjestelmän tulevaa toiminnallisuutta. Tämä mahdollistaa integraatiotestauksen kulkevan jopa komponentin kehityksen edellä. Näin integraatiotestejä voidaan kehittää samaan aikaan, kun liitettävää komponenttia vielä kehitetään, jolloin palaute komponentin toimivuudesta saadaan heti, kun liitettävän komponentin kehitys valmistuu. (2, s. 50.)

3.4 Järjestelmätestaus

Integraatiotestauksen jälkeen on seuraavana vuorossa järjestelmätestaus, jonka tarkoitus on varmistaa, että järjestelmä kokonaisuutena täyttää sille asetetut tekniset vaatimukset. Moni järjestelmän toiminnallisuuksista ja ominaisuuksista tulee vasta kunnolla käytettäväksi, kun järjestelmä toimii kokonaisuutena tuotantoa vastaavassa ympäristössä. Järjestelmätestaus tulisi aina suorittaa tuotantoa mahdollisimman tarkasti jäljittelevässä ympäristössä, koska näin saadaan mahdollisesta ympäristöstä johtuvat virheet kiinni. Kaikista yleisimmät virheet, joita ilmenee järjestelmätestauksessa, ovat oikeuksien puuttuminen. Kun ohjelmisto on siirretty tuotantoa vastaavaan ympäristöön, on sillä yleensä paljon tiukemmat turvamäärityksen kuin kehityksen aikana. Tämän takia onkin hyvin todennäköistä, että vielä järjestelmätestausvaiheessa havaitaan virheitä, jotka saattavat aiheuttaa vahinkoa asiakkaan järjestelmiin tai toimintaan. Järjestelmätestausta voisikin kuvailla ohjelman teknisenä kenraaliharjoituksena, joka varmistaa, että kaikki toimii niin kuin on suunniteltu. (2, s. 54.)

Järjestelmätestaus on kaikista laajin testauksen taso, ja se voidaan jakaa pienempiin osa-alueisiin, jotka keskittyvät aina tiettyyn järjestelmän ominaisuuteen. Tärkeimmät osa-alueet ovat järjestelmän toiminnallisuus, tietoturva ja suorituskyky. Järjestelmätestaus tapahtuu yleensä toteuttamalla päästä päähän -testausta, jossa testitapauksina toimivat kokonaiset käyttötarkoitukset. Äkkiseltään järjestelmätestaus saattaa kuulostaa hyvin samankaltaiselta kuin integraatiotestauksen viimeiset vaiheet, jossa testataan järjestelmän osien toimivuutta yhdessä, mutta suurin ero syntyy toteutustavassa. Järjestelmätestaus suoritetaan simuloimalla järjestelmän käyttöä täysin käyttöliittymän kautta, kun integraatiotestaus puolestaan keskittyy täysin rajapintojen väliseen viestittelyyn, joka tapahtuu koodia suorittamalla ilman käyttöliittymää.

3.5 Hyväksymistestaus

Kaikki tähän mennessä olleet testauksen tasot ovat kuuluneen suurimmaksi osaksi kehittäjän vastuualueelle, ja ne on suoritettu ennen kuin tuotetta on esitelty asiakkaalle. Ennen kuin tuote voidaan viimein ottaa käyttöön, on suoritettava vielä yksi testauksen taso eli hyväksymistestaus. Hyväksymistestauksessa keskitytään tarkastamaan tuotetta asiakkaan näkökulmasta. Hyväksymistestaus on ainut testauksen osa-alue, jossa lähes aina asiakas tai asiakkaan edustaja on tuotteen loppukäyttäjä kanssa mukana. Hyväksymistestauksen tarkoitus on varmistaa, että tuote täyttää kaikki sopimuksessa sovitut asiat ja voidaan todeta valmiiksi. Testauskriteerit määräytyvät hyvin pitkälti sopimuksessa laadittujen vaatimuksien mukaan. Myös lailliset vaatimukset tarkistetaan viimeistään tässä vaiheessa, kuten salaiseksi luokitellun materiaalin turvallinen käyttö. Yleinen toimintatapa on, että kehitystiimistä erillinen testaustiimi suorittaa hyväksyntätestauksen ja esittää sen tuloksineen asiakkaalle. Väärinymmärrysten välttämiseksi on hyvin tärkeää, että hyväksymiskriteerit on luotu läheisessä yhteistyössä asiakkaan kanssa. Hyväksymistestaus tulisi suorittaa asiakkaan omassa ympäristössä, mutta ei kumminkaan vielä lopullisessa tuotantoympäristössä, riskien minimoimiseksi. Ympäristömuutokset voivat taas tuoda esiin uusia ongelmia.

Hyväksymistestauksen keskeisin osa on testata tuote käyttäjän näkökulmasta. Tässä korostuu eri käyttäjäryhmien tarpeet ja odotukset tuotteelle. Jos yksikin käyttäjäryhmä kokee tuotteen käytön epämiellyttäväksi tai puutteelliseksi, voi tämä estää tuotteen julkaisemisen ennen kuin ongelma on ratkaistu. Tämä saattaa tapahtua, vaikka tuote on teknisten ominaisuuksien osalta kunnossa, mutta esimerkiksi tuotteen käyttöliittymä on

liian epäselvä käyttäjän näkökulmasta ja hankaloittaa näin tuotteen käyttöä. Testaus käyttäjän näkökulmasta on tärkeää tuotteen omistajan kannalta, koska tuotetta myydessä on tuotteen omistajan maine ja liiketoiminta panoksena.

Hyväksymistestauksen ensimmäinen vaihe on toimintakuntoinen hyväksyntätestaus, jossa ohjelman toiminta varmistetaan sen hallinnoijan näkökulmasta. Ohjelmasta riippuen tähän voi sisältyä datan varmuuskopiointi ja palautusominaisuuksiin liittyviä asioita. Myös käyttäjien hallinta ja ylläpidettävyys on keskeinen tarkastelun kohde. Jos tuotteen on tarkoitus toimia monissa eri ympäristöissä, kuten loppukäyttäjien tietokoneella, on hyvin kallista ja mahdotonta suorittaa testaus kaikissa mahdollisissa ympäristöissä. Tätä varten tuotteen omistaja voi suorittaa niin sanotun kenttäkokeen, jonka tarkoitus on kartoittaa käyttäjien ympäristöissä aiheutuvia ongelmia. Tämä tapahtuu julkaisemalla viimeisin ja vakain versio ohjelmasta, ja se annetaan käytettäväksi loppukäyttäjille. Tämä voidaan suorittaa ohjeistamalla loppukäyttäjälle, kuinka ohjelman pitäisi toimia. Loppukäyttäjän tehtäväksi jää kokeilla tuotteen käyttämistä ja antaa palautetta, kuinka hyvin käyttäminen onnistui. Toinen vaihtoehto on antaa loppukäyttäjän vain käyttää ohjelmaa ilman ohjeistusta, jolloin nähdään, missä kohdissa käyttäjälle ilmenee ongelmia hahmottaa tuotteen toimintaa tai sen kulkua. Näin saadaan käyttäjältä arvokasta tietoa tuotteen ongelmista ja sen aiheuttamista mielipiteitä. Yleensä tällainen käyttäjiä läheinen testaus tapahtuu kahdessa vaiheessa, joita kutsutaan nimillä alfa ja beta. Alpha on näistä ensimmäinen ja tapahtuu yleensä tuotteen omistajan tiloissa vain hyvin pienen siihen valitut käyttäjä ryhmän suorittamana. Kun vuorostaan beta-vaiheessa tuote julkaistaan kaikkien halukkaiden käyttöön isomman otannan saamiseksi. Tietysti tuotteen ennakoidusta suosiosta riippuen beta-versioiden maksimimäärää voidaan rajoittaa mahdollisten suorituskyky ongelmien vuoksi laadukkaamman palautteen saamiseksi.

3.6 Regressiotestaus

Vaikka tuote on valmistunut, ja se on käyttäjien jatkuvassa käytössä, testaus sen osalta ei ole vielä ohi, vaikka niin voisi luulla. Tästä alkaa vasta tuotteen uusi vaihe sen elinkaaressa, johon sisältyy tuotteen ylläpitoa sekä ominaisuuksien päivittämistä ja parantamista. Ohjelmistotkin kärsivät ajan vaikutuksesta ja näin vaativat päivittämistä. Aina kun uusi päivitys lisätään tuotteeseen, on se syytä testata samalla tavalla kuin muutkin sen osat kehitystyön aikana on testattu. Tämän lisäksi on syytä suorittaa uudelleentestaus, jota yleisemmin kutustaan regressiotestaukseksi. Regressiotestauksessa vanhat osat

testataan uudelleen sen varalta, ettei päivityksen yhteydessä uusi ohjelmiston osa tai osan muutos ole aiheuttanut virhettä. Regressiotestaus kohdistuu vanhoihin jo testattuihin osiin varmistamaan, ettei sivuvaikutuksina syntynyt uusia virheitä tai vanhoja ennen huomaamattomia ole noussut uudestaan esille. Regressiotestausta voidaan suorittaa kaikilla edellä mainituilla testauksen osa-alueilla. Regressiotestauksessa käytettävät testitapaukset tulee olla hyvin dokumentoituja, ettei epävarmoja tilanteita pääse syntymään tai käsitys toiminallisuudesta muutu ajan kuluessa. Myös testitapausten uudelleenkäytettävyys on erittäin tärkeää, että saadaan aina täysi varmuus, etteivät toiminnot ole muuttuneet ei-toivotulla tavalla. Regressiotestauksessa voidaan hyödyntää jo kehitysvaiheessa luotuja testitapauksia. On erittäin tärkeää kartoittaa, kuinka kattavaa regressiotestauksen tulee olla, koska koko järjestelmän uudelleen testaus voi koitua usein turhaan kalliiksi. Siksi riski ja testauksesta saatavaa hyötysuhdetta on syytä arvioida jatkuvasti jokaisen päivityksen kohdalla. Tästä syystä dokumentaatio nousee tärkeään rooliin, koska se antaa arvokasta tietoa ja helpottaa testauksen rajausta. Uudelleentestaus kannattaa keskittää kaikista vikaherkimpiin ja päivityksen kannalta keskeisimpiin osiin. Koska regressiotestauksessa käytettävät testitapaukset tullaan suorittamaan tuotteen elinajan aikana useaan kertaan, on tämä hyvä tilaisuus hyödyntää automatisaatiosta saatavia etuja.

4 Staattiset testimenetelmät

4.1 Mitä ovat staattiset testausmenetelmät

Staattiset testimenetelmät ovat usein hyvin aliarvostettuja. Toisin kuin dynaamisissa testauksessa staattiseen testaukseen ei kuulu koodin tai ohjelman suorittamista vaan se analysoidaan. Tämän voi suorittaa joko muiden ihmisten kanssa tai käyttämällä tähän tarkoitettuja työkaluja. Mikä tahansa dokumentti tai tiedosto on mahdollista tarkastaa staattisilla menetelmillä. Staattisten testausmenetelmien tarkoitus on löytää virheitä määritellyistä vaatimuksista ja projektisuunnitelmista. Staattisista menetelmistä saadulla palautteella pyritään välttämään virheiden tekoa kehitysvaiheessa ja optimoimaan sen kulkua. Päätarkoitus on estää virheiden tapahtuminen, koska virheet ovat helpoin ja halvin korjata heti suunnitteluvaiheessa.

Katselmoinnit kohdistavat tarkasteltavan dokumentit ihmisen analyyttiselle päättelykyvylle. Tämä tapahtuu dokumentin sisällön tarkalla läpikäynnillä ja yrittämällä ymmärtää

monimutkaiset dokumentit, jotka ovat tarkastelun kohteena. Katselmointeja on eri tasoisia. Ne luokitellaan niiden tarkkuuden, virallisuuden, saatavilla olevien resurssien ja tarkasteltavan kohteen mukaan. Katselmointi on yleinen käsite kaikille ihmisen suorittamille staattisille tarkastusmenetelmille. Katselmoinnit ovat tehokas tapa parantaa dokumenttien laatua. Katselmoinnit tulisi suorittaa heti, kun dokumentti on luotu, että mahdolliset virheet sekä epäjohdonmukaiset asiat löydetään heti alussa ennen kuin ne pääsevät edes muodostumaan osaksi tuotetta. Virheiden poistaminen nostaa laadun määrää ja selkeät virheettömät dokumentit selkeyttävät koko projektin toteuttamista ja kehitystyö kohdistuu oikeisiin asioihin. Aikainen virheiden havainnointi mahdollistaa nopean ja kustannustehokkaan korjauksen, kun uudelleen tekemisen määrä minimoituu. Dokumentaation selkeys nopeuttaa myös kehitystyötä, kun siinä esiintyy vähemmän ongelmia. Yleensä ihmiset saattavat kokea katselmoinnit ylimääräisenä työnä, koska niistä saatava hyöty tulee vasta pitkässä juoksussa hiljalleen. Katselmointien toinen ongelma on myös se, että jotkut ihmiset voivat kokea dokumentin luojana itse olevansa tarkastelun kohteena dokumentin sijaan. Näissä tilanteissa on hyvin tärkeää painottaa neutraalisuutta, eikä keskittyä siihen, kenen tekemä virhe oli. (1, s.59, 60.)

4.1.1 Läpikäynti

Läpikäynti on epävirallinen katselmointimuoto, jonka tarkoituksena on löytää virheitä, monitulkinnallisia kohtia ja muita dokumentissa mahdollisesti esiintyviä ongelmia. Läpikäynnissä dokumentin tuottaja esittelee dokumentin vaiheet loogisessa järjestyksessä läpikäyntitilaisuuteen tulleille ihmisille. Kuuntelijoiden tehtävänä on yrittää huomata mahdolliset virheet ja epäselkeät kohdat. Kun kuuntelija huomaa ongelman tai on epävarma dokumentissa esitettävästä asiasta, on hänen tehtävä keskeyttää esittäjä ja nostaa tämä kohta puheenaiheeksi ratkaisun löytämiseksi. Ongelman korjattua tai sen ratkaisun ylös kirjoitettua dokumentin läpikäynti voi jatkua. Läpikäynnit toimivat parhaiten pienissä alle kymmenen hengen kehitystiimeissä. Tällöin läpikäynnit ovat tehokkaita eivätkä kuluta liikaa resursseja. Läpikäynnin ollessa epävirallinen katselmointimuoto, sopii se vain eikriittisille dokumenteille. Läpikäynnin yhteydessä voidaan myös varmistaa, että dokumentin kannalta tärkeät ihmiset tulevat tietoiseksi sen sisällöstä ja ymmärtävät sen asiasisällön oikein. Läpikäynteihin voi antaa kuulijoille dokumentin etukäteen tarkasteltavaksi sujuvamman tapaamisen toteuttamiseksi, mutta se ei ole välttämätöntä. (2, s. 83.)

4.1.2 Tarkistus

Tarkastus on katselmoinneista kaikista virallisin muoto. Se seuraa virallista ennalta määrättyä rakennetta. Jokaisella tarkastuksen osanottajalla on oma rooli hoidettavana. Myös tarkastuksen aloittamiselle, kululle ja päättymiselle luodaan tarkat ennalta määritetyt ehdot. Tarkoitus on löytää mahdolliset virheet, määrittää dokumentin laatu ja parantaa dokumentin ja kehitystyön laatua. Tarkastajilla on oma tehtävälistansa valmistautuakseen tarkastukseen ennen tarkastuksen alkua. Tarkastus seuraa seuraavanlaista rakennetta. Tarkastuksen puheenjohtaja ohjaa tapahtumaa ja aloittaa sen esittelemällä siihen osallistuvat henkilöt ja heidän roolinsa. Myös lyhyt kertaus tarkastelun tarkoituksesta ja aiheesta kuuluu alkupuheeseen ja samalla varmistaa, että kaikki ovat valmistautuneet tarpeen mukaan tarkastukseen. Alkupuheen aikana on myös mahdollista keskustella, kuinka paljon aikaa kukin on kerennyt käyttää valmistautumiseen ja paljonko virheitä on löydetty etukäteen. Esiin tulleet ongelmat kirjataan ylös myöhempää käsittelyä varten. Niin kuin läpikäynnissä tarkastuksessakin katselmoijan tehtävänä on esittää kysymyksiä ja esittäjä pyrkii vastaamaan niihin. (2, s. 84.)

Puheenjohtajan rooliin kuuluu keskustelun rajaaminen ja aikataulusta huolehtiminen kuten myös pitää huoli, että kaikki suunnitellut osiot ehditään käydä läpi. Jos keskustelussa ei päästä nopeasti yksimielisyyteen, on puheenjohtajan vastuulla katkaista keskustelu, jotta aikataulussa pysyttäisiin. Puheenjohtaja pitää myös huolen, että kirjanpito pysyy samassa tahdissa keskustelun kanssa. (2, s. 84.)

Tarkastuksen lopuksi kaikki kirjanpito käydään vielä läpi ja varmistetaan, että se kattaa tarkastuksen koko kulun. Samalla voidaan keskustella tarkastuksen aika esiin nousseista erimielisyyksistä ja pyrkiä ratkaisemaan ongelmat. Jos ratkaisuun ei päästä, tästä tulee jäädä merkintä kirjanpitoon. Tarkastuksen viimeinen vaihe on päättää, päästiinkö tarkastuksessa yksimielisyyteen vai vaatiko se lisätyötä ja mahdollisen uudelleen tarkastuksen. Tarkastuksen yhteydessä kerätään myös tietoa sen kulusta ja saadusta hyödystä kehitysprosessin parantamiseksi. Näin ollen koko prosessi hyötyy tarkastuksesta eikä vain yksittäinen dokumentti. Tarkastus on kuitenkin hyvin raskas prosessina muihin katselmointimuotoihin verrattuna, joten sen käytölle tulee olla hyvät perustelut ja tulisi vain suorittaa toiminnan kannalta kaikista kriittisimpiin dokumentteihin. (2, s. 85.)

4.1.3 Tekninen katselmointi

Tekninen katselmointi keskittyy katselmoimaan dokumentissa esitettyä asiaa teknisestä näkökulmasta. Katselmoijien tulee olla aiheeseen teknisesti päteviä, yleensä alansa asiantuntijoita. Projektin hallintopuolen ihmiset eivät yleensä osallistu teknisiin katselmointeihin, vaan se jää täysin teknisten asiantuntijoiden vastuulle. Katselmoijat kirjoittavat ylös kommentteja etukäteen ja antavat ne katselmoinnin pitävälle henkilölle ennen tilaisuutta. Tilaisuuden järjestäjä käy läpi katselmoijien löydökset ja asettaa ne tärkeysjärjestykseen. Itse tilaisuuden tarkoitukseksi jää tärkeimpien korjausten käsitteleminen ja ratkaisu. Tapaamisen aikana kaikki ongelmat kirjataan ylös lopulliseen raporttiin. Isoin osa työstä koostuu valmisteluvaiheesta, ja jokainen voi tehdä valmistelun heille sopivalla ajalla. Näin yhteisen ajan löytäminen on helpompaa ja vältetään pitkiä monen hengen tapaamisia. Teknisen katselmoinnin tulos tulee olla yksimielinen, ja kaikki erimielisyydet tulee kirjata ylös. Katselmoijien tarkoitus ei kuitenkaan ole tehdä päätöstä ongelmien ratkaisusta vaan antaa siitä asiantuntijan näkökulma, jonka pohjalta projektin hallinto tekee päätöksen. Teknisen katselmoinnin virallisuus riippuu hyvin pitkälti sen aiheesta ja erittäin kriittisten valintojen kohdalla siinä voidaan käyttää tarkastuksen tapaista formaattia, mutta tekniset katselmoinnit voivat olla myös hyvin epävirallisia muutaman henkilön tapaamisia. (2, s. 85.)

4.1.4 Epävirallinen katselmointi

Epävirallinen katselmointi on kevyin katselmoinnin muoto. Epävirallisessa katselmoinnissa dokumentin laatija käynnistää katselmoinnin pyytämällä katselmoijia suorittamaan katselmoinnin ja antamaan siitä palautetta. Yleensä epäviralliseen katselmointiin ei liity tapaamisia vaan kukin suorittaa sen omalla ajallaan. Yleensä katselmointi tapahtuu toisen saman asian parissa työskentelevän kollegan toimesta. Tarkoitus on löytää yksinkertaiset ajattelu- tai huolimattomuusvirheet. Virheitä ei dokumentoida sen tarkemmin, vaan yksinkertaiset kommentit riittävät niiden korjaamiseksi. Epävirrallinen katselmointi on kaikista käytetyin katselmointi muoto sen kevyen ja nopean rakenteen puolesta. Kehitys työn yhteydessä epävirallista katselmointia käytetään hyvin paljon esimerkiksi uusien koodikomponenttien kohdalla. Koodin tuottanut kehittäjä voi pyytää projektin toista kehittäjää antamaan oman mielipiteensä koodin toteutuksesta ennen sen liittämistä isompaan kokonaisuuteen. (2, s. 86.)

4.2 Staattiset työkalut

Esimerkiksi automaattinen kirjoituksen korjaus dokumentointiohjelmissa luokitellaan staattiseksi työkaluksi. Siinä ohjelma tarkastaa käyttäjän kirjoittaman tekstin kirjoitusvirheiltä. Staattiset työkalut prosessoivat käyttäjän luoman tekstin tai koodin, mutta ei suorita sitä. Tarkoituksena on vaan parantaa dokumentin luettavuutta. Staattisia työkaluja käytettäessä dokumentin tulee seurata määrättyä rakennetta, jotta staattinen työkalu pystyy käsittelemään sitä. Kaikissa projekteissa on yleensä käytössä jonkin tason staattinen työkalu, esimerkiksi Word-dokumentin oikeinkirjoituksen tarkistus. Ohjelmointiprojekteissa isoin hyöty staattisista työkaluista tulee, kun niitä käytetään tuotetun koodin tarkistamiseen. Tarkistaminen tapahtuu yleensä kehityksen aikana tai viimeistään komponentti testauksen yhteydessä. Tässä tarkistuksessa voidaan tarkastella koodin oikein kirjoitusta samaan tapaan kuin tavallistakin tekstidokumenttia. Staattisilla työkaluilla voidaan myös asettaa koodin muotoiluun liittyviä asetuksia, kuten rivin maksimipituus, tyhjien välilyöntien ja rivien asetuksia, jotta koodi pysyisi mahdollisimman siistinä ja helppolukuisena. (1, s. 68.)

Kun staattisissa työkaluissa on paljon yksityiskohtaisia tarkistuksia, on yleensä sen antama raportti myös pitkä ja täynnä varoituksia ja virheitä, mutta tämä on välttämätöntä laadukkaan koodin tuottamiselle. Kun varoituksia ilmenee paljon, on ne syytä luokitella, jotta niiden läpikäyminen pysyisi tehokkaana. Tämän voi tehdä jakamalla virheilmoitukset virheisiin, jotka on pakko korjata ja varoituksiin, jotka toimivat enemmän vain huomautuksina eivätkä vaadi pakotettua korjausta.

Staattisista työkaluista saatu hyöty ei rajoitu vain oikeinkirjoitukseen, vaan niillä voidaan myös analysoida ohjelmointi käytäntöjen toteutusta, kuten muuttujien määrittelyä ja niiden kulkua koodin läpi. Tätä kutsutaan datan läpikulkutarkastukseksi. Muuttujien osalta staattiset työkalut pitävät huolen, että, niiden muoto pysyy oikeana. Esimerkiksi muuttuja, joka on alustettu muuttuvaksi, mutta sen arvoa ei muuteta koodin suorituksessa missään vaiheessa, niin staattisilla työkaluilla saadaan siitä virheilmoitus, joka kertoo, että muuttujan tulisi olla staattinen. Näin koodista saadaan turvallisempaa, kun tarpeeton muuttujien muokkaaminen saadaan estettyä. Tämänkaltainen tilanne ei vielä yksinään aiheuta virhetilannetta suoritusvaiheessa, joten sitä ei voida havaita dynaamisilla testaus menetelmillä, mutta ilman staattisia tarkastuksia se jäisi riskiksi jatkokehitykselle. Yleensä jokainen koodin suorittamiseen käytettävä työkalu sisältää vähintään koodin

oikeinkirjoituksen tarkistuksen, joka suoritetaan ennen koodin suorittamista kaatumisien välttämiseksi. (1, s. 68, 69.)

5 Hyvän testaus kokonaisuuden toteuttaminen projektissa testauksen suunnitelma

5.1 Testaussuunnitelma

Testisuunnitelman luominen on testipäällikön tärkein tehtävä. Sillä varmistetaan, että testaajilla on lista tehtäviä ja maaleja, joiden pohjalta voidaan seurata testaamisen etenemistä ja arvioida sen kokonaisuutta. Testisuunnitelma tulisi olla käytössä kaikissa ohjelmistoprojektin kehitysvaiheissa. Testauksen pohjana tulisi aina olla projektin testaussuunnitelma. Projektin testaussuunnitelmassa määritetään yleisellä tasolla testaukseen liittyvät toimet, mitä testitasoja projektissa on ja aikataulullisia asioita. Projektikohtainen testisuunnitelma kuuluu testauksen ensimmäiseen vaiheeseen ja tulisi näin tehdä heti projektin alussa, jotta testaus pystytään aloittamaan rinnakkain kehitys työn kanssa. Pelkkä projektikohtainen testisuunnitelma ei ole kuitenkaan tarpeeksi tarkka, vaan jokaisella projektintestisuunnitelmassa mainitulle testauksen osa-alueelle, kuten esimerkiksi hyväksymistestaukselle, on tehtävä oma kyseisen testaustason kattava suunnitelma, jossa tarkasti määritellään, mitä testataan ja miten. (13.)

Yksi testisuunnitelman tärkeistä osista ovat aloituskriteerit. Aloituskriteereissä määritellään milloin minkäkin vaiheen testaus voi alkaa, jotta se olisi tehokasta ja tuottavaa. Tyypillisiä testauksen aloituskriteereitä ovat: testaus ympäristön valmius toteutusta varten, tarvittavat työkalut on asennettu, testattava tuote on saatavilla, testaukseen mahdollisesti tarvittava data on saatavilla ja oikeanlaista sekä tietysti, että testauksen suunnitelma on toteutettu. (13.)

Tietysti kun testauksen aloittamiselle on omat kriteerit, niin myös sen lopettamiselle on omat kriteerit. Lopetuskriteereitä käytetään hahmottamaan, milloin kyseinen testaustyö on saatu valmiiksi tai milloin sen tulisi loppua. Niin kuin aloituskriteerit niin myös lopetuskriteerit tulisi luoda jokaiselle testauksen tasolle, ettei testaustyö veny loputtomiin eikä resursseja käytettäisi turhaan. Tyypillisiä lopetuskriteereitä ovat, että kaikki suunnitellut testitapaukset on suoritettu ja haluttu testikattavuus saavutettu, kaikki testauksessa löydetyt merkittävät virheet on korjattu tai raportoitu ja kaikki kriittiset osa-alueet on testattu

huolellisesti. Myös budjetin kuluminen loppuun voi olla lopetuskriteerinä joissakin projekteissa. Yleensä näissä tapauksissa testausta suoritetaan prioriteettijärjestyksessä, kunnes testaukselle varattu budjetti on loppunut ja tämän jälkeen tehdään uudelleenarviointi, jossa määritellään, onko saavutettu testikattavuus riittävä vai tarvitseeko se lisärahoitusta. Lopetuskriteerit tulisi luoda ennen testauksen aloittamista, ettei mahdolliset aikataulupaineet testauksen aikana pääse vaikuttamaan päätökseen testauksen valmiudesta. (13.)

5.2 Testausympäristöt

Jokaisessa ohjelmistoprojektissa kannattaisi olla vähintään pari erilaista testausympäristöä. Erilaiset testausympäristöt helpottavat ja nopeuttavat kehitystyötä ja testausta. Tavallisin ympäristö, mikä jokaisella kehittäjällä ja testaajalla tulisi olla, on lokaali ympäristö. Lokaalin ympäristön etuihin kuuluu, että se on erittäin nopea ja kehittäjällä on yleensä täyden oikeudet muokata ja kokeilla ympäristön sisällä mitä vain pienellä vaivalla. Lokaaliympäristö on erittäin hyödyllinen etenkin kehityksen ja testauksen alkuvaiheessa, kun halutaan varmistaa ominaisuuksien toimintaa mahdollisimman erilaisissa tilanteissa. Näitä tilanteita on kaikista helpoin toteuttaa, kun rajoitteita on mahdollisimman vähän. Lokaaliympäristö soveltuukin erittäin hyvin myös matalantason automaatiotestauksen, kuten yksikkö- ja integraatiotestaukseen. Kun yksikkö- ja integraatiotestaus toteutetaan aina kehitystyön päätteeksi, ennen koodin liittämistä yhteiseen projektiin, pystytään näin karsimaan jo suurin osa ohjelmistovirheistä pois ennen kuin ne pääsevät vaikuttamaan muun projektin työskentelyyn. (5.)

Lokaalia ympäristöä astetta ylempänä on yleensä niin sanottu alfa-ympäristö. Alfa-ympäristö on yleensä kaikkien saatavilla oleva hiekkalaatikko tyylinen ympäristö. Alfa-ympäristön tarkoitus on olla matalankynnyksen yhteinen ympäristö, jossa voidaan helposti toimia yhdessä koko projektin voimin ilman pelkoa virheiden aiheuttamista haitoista. Alfa-ympäristö soveltuukin hyvin uusien ominaisuuksien esittelyyn muulle kehitystiimille.

Viimeisenä ympäristönä, joka jokaisesta projektista yleensä löytyy, on tuotannon testiympäristö. Tuotannon testiympäristö pyrkii jäljittelemään tuotannon olosuhteita mahdollisimman tarkasti ja näin ollen soveltuu parhaiten järjestelmä- ja hyväksyntätestaukseen. Tuotannon testiympäristö on yleensä se ympäristö, missä testaustiimi tekee suurimman osan työstään, kun aikaisemmat kaksi matalan tason testiympäristöä ovat

enemmän kehitystiimin vastuulla. Tuotannon testiympäristöä voidaan myös hyödyntää tuotteen esittelyssä sen tilanneelle taholle, kuten myös ensimmäisten oikeiden käyttäjäkokemusten hankkimiseen yleisöltä. (5.)

5.3 Testiautomaatio

Testiautomaatio on hyvin suosittu tapa tehostaa testauksen toimintaa, mutta sen käyttöönotto on suunniteltava tarkasti, että siitä saadaan haluttu hyöty irti. Huonosti toteutettu testiautomaatio voi pahimmassa tapauksessa aiheuttaa vain ylimääräistä työmäärä ilman suurempia hyötyjä. Testauksen automatisointia suunnitellessa tulee ottaa huomioon testattavan kohteen tarpeet, minkälaisia testitapauksia halutaan automatisoida ja mikä testiautomaatio työkalu sopii kyseiseen käyttötarkoitukseen parhaiten. Turvallisin tapa ottaa mikä tahansa työkalu käyttöön on ensiksi luoda siitä mahdollisimman pieni soveltuvuusselvitys. Soveltuvuusselvityksessä työkalu otetaan koekäyttöön vain pienessä osassa projektia, jotta voidaan kerätä tarkempaa tietoa sen soveltuvuudesta haluttuun käyttötarkoitukseen. On myös hyvin tärkeää pitää mielessä, että kaikkea ei aina kannata automatisoida tai ole edes mahdollista automatisoida. Paras hyöty automaatiosta saadaan silloin, kun automatisoidaan useasti toistettavia testitapauksia tai ihmiselle mahdottomia tapauksia, kuten suorituskykytestaus.

5.3.1 Automatisoinnin vahvuudet

Selkein automatisoinnin vahvuus on sen kyky suorittaa testitapauksia, joihin ihminen ei manuaalisesti pystyisi tai ei olisi resurssien käytön kannalta järkevää suorittaa manuaalisesti, koska testiautomaatiolla ne voidaan suorittaa murto-osassa tästä ajasta. Testiautomaation tärkeys korostuu sitä enemmän, mitä ketterämpi ohjelmistokehitysprojekti on. Tämä johtuu ketterän kehityksen luonteesta edetä pienin askelin eteenpäin, mutta askelia otetaan usein jopa viikoittain, jolloin testaukselle syntyy paljon toistoa. Vaikka testiautomaatioon toimivaksi saamiseen voi aluksi kulua monta kertaa enemmän aikaa kuin yksittäisen ominaisuuden testaamiseen, voittaa se pitkässä juoksussa toistuvaan manuaalitestauksen kuluvan ajan. (12.)

Toinen automatisoinnin vahvuuksista on vähentää inhimilliset virheiden määrä ja näin luoda lisää luotettavuutta. Hyvin toteutettu testiautomaatio on selkeää ja suoraviivaista. Tällöin taataan, että testitapaukset suoritetaan aina tismalleen samalla tavalla jokainen

kerta väsymättä toiston määrästä riippumatta. Tämän hyödyn huomaa parhaiten kuvittelemalla ihmistä tilanteeseen, jossa hän joutuu toistamaan samoja testitapauksia manuaalisesti viikosta toiseen. Tällöin ihmisen suoritteessa saattaa tapahtua suurtakin variaatiota testauksen laadun kannalta kiireestä ja toiston aiheuttamasta kyllästymisestä johtuen, kun kulmia aloitetaan oikomaan. (12.)

Monien testiautomaatiotyökalujen mukana tulee myös automaattinen dokumentointi testien tuloksista. Nämä dokumentit yleensä sisältävät erittäin tarkan ja kattavan vaihe vaiheelta -kuvauksen suoritetusta testitapauksesta. Kehittyneimmissä työkaluissa on jälkeen päin mahdollista katsoa kuvakaappauksista tai jopa suoraan nauhoitetulta videolta, missä kohtaa käyttöliittymässä virhe on tapahtunut. Selkeästä ja kattavasta tulosten raportoinnista on suuri hyöty kehitystyölle virhetilanteita selvittäessä, jotta nopea ongelman paikannus ja korjaus voi tapahtua. Testausraportit toimivat myös hyvänä tietolähteenä ohjelmiston valmiudesta tuotteenomistajille, jotka haluavat seurata ohjelmiston kehitystä tuotteen julkaisuun liittyen.

5.3.2 Automatisoinnin heikkoudet

Automatisointia voi ajatella pienenä erillisenä projektina, ja niin kuin kaikissa projekteissa, suurin riski on sen täydellisessä epäonnistumisessa. Automatisoinnin epäonnistuessa on siihen kulunut paljon aikaa ja resursseja ilman merkittävää hyötyä ja näin ollen voi olla vaikea päästä huonosti toimivasta ratkaisusta irti, kun siihen on jo sijoitettu paljon työtä. Siksi on tärkeää automaatiota toteuttaessa pitää tavoitteet selkeänä ja realistisina ja tarkkailla niiden toteutumista koko kehityksen ajan. (3.)

Automaattisten testien oletetaan hyvin usein löytävän paljon virheitä, mutta tämä ei aina pidä paikkaansa. Lähes aina, kun testiautomaatiota aloitetaan kehittämään, ovat testattavat ominaisuudet jo manuaalisesti testattu tätä ennen, että tiedetään, voiko testien kehitys alkaa. Näin ollen, kun automaattiset testit valmistuvat, menevät ne oletetusti läpi, koska ohjelmisto on sillä hetkellä toimiva. Tämä saattaa johtaa tunteeseen, että testeistä ei ole mitään hyötyä, koska ne eivät saa virheitä kiinni. Pitää kumminkin muistaa, että testiautomaation vahvuus ilmenee vasta pitkässä juoksussa, kun regression myötä vanhoihin ominaisuuksiin ilmaantuu uusia virheitä. Testien vihreään väriin ei saa kumminkaan sokaistua, sillä täytyy muistaa, mitä se todellisuudessa tarkoittaa. Testien vihreinä ilmoittama läpäisty merkintä ei automaattisesti tarkoita, etteikö virheitä voisi olla, vaan todellisuudessa se tarkoittaa, että testisuoritus onnistui ilman oletettuja ongelmia. Jos

testiä ei ole tehty huolellisesti siten, että se saa kaikki halutut virhetilanteet kiinni, on mahdollista, että se palauttaa virheellisen vihreän vastauksen. Testiautomaation kannalta ei ole mitään niin vaikeasti huomattavaa kuin virheellisesti näytettävä läpäisty merkintä, koska erittäin harvoin läpäistyksi merkittyjä testejä tullaan tutkimaan. Tämän takia testiautomaation toteuttaminen vaatii korkeaa ammattitaitoa ja tarkkaavaisuutta, mitä ei ole aina helppo löytää. (4.)

Samaan tapaan kuin vireellinen läpäisty merkintä aiheuttaa ongelmia, niin sama pätee myös virheelliseen hylätty-merkintään. Virheellinen hylätty-merkintä on huomattavasti helpompi huomata kuin virheellinen läpäisty-merkintä, koska punainen väri kiinnittää lähes aina testaajan tai kehittäjän huomion. On kuitenkin mahdollista, että testien luotettavuus laskee niin huonolle tasolle, että punainen väri menettää sen huomion herättävän merkityksen, kun oletukseksi muodostuu sen olevan virheellisesti punaisella. Tästä voi pahimmassa tapauksessa seurata se, että oikeatkin virhetilanteet ohitetaan tämän oletuksen mukana. Tämän takia onkin erittäin tärkeää saada testit toimimaan luotettavasti, että niistä saataisiin hyötyä jarruttavan taakan sijaan. (6.)

Vaikka testiautomaation käyttöönotto on onnistunut onnistuneesti, saattaa ajan kuluessa ilmestyä uusia ongelmia. Projektin edetessä saattaa testien koko ja lukumäärä nousta hyvinkin suureksi. Mitä isompi automaatiotestauksen kokonaisuus on, sitä enemmän aikaa ja resursseja sen ylläpito tulee vaatimaan. Kuten aina mitä isompi kokonaisuus sitä vaikeampi sitä on hallita ja säilyttää tasaisen hyvä laatu, joten testitiimin vastuulle usein jää tilanteen seuraaminen. Kun huomataan, että resurssit eivät enää riitä tilanteen ylläpitämiseen, pitää testitiimin reagoida tähän ja ilmoittaa siitä johtavalle tasolle, että korjaaviin toimenpiteisiin voidaan ryhtyä. Aina resurssipulan korjaaminen ei ole helppoa tai nopeaa, joten ennakoivasta toiminnasta on aina näissä tilanteissa hyötyä. (3.)

5.4 Raportointi

Varmistaakseen luotettavan ja nopean virheiden korjaamisen projektissa hyvin toimiva prosessi eri testi tasoilta saatavasta palautteesta on ehdottoman tärkeää. Jokaisen suoritetun testin jälkeen on sen tulokset tarkistettava. Saatuja tuloksia verrataan oletettuihin tuloksiin ja jokaisesta poikkeamasta, joita ilmeni testauksen aikana, on syytä katsoa tarkemmin virheen varalta. Kun poikkeama testituloksissa huomataan, testaajan tulee varmistaa, että virhe johtuu oikeasti ohjelmiston viallisesta toiminnasta, eikä testin virheestä,

koska testeissäkin saattaa olla virheitä ja puutteita samalla lailla kuin kehitettävässä ohjelmistossa. Jos ongelma johtui ohjelmiston viallisesta toiminnasta, on ongelma raportoitava. Jos taas vika oli itse testissä, on se syytä korjata ja suorittaa testi uudestaan. Ongelma tulee raportoida siten, että sen korjaavalla kehittäjällä on kaikki tarvittava tieto ja toimenpiteet ongelmatilanteen uudelleen luomiseen, jotta kehittäjä voi alkaa selvittämään, mistä ongelmassa on kyse. (11.)

Yleisesti jokaisessa projektissa tulisi olla arkisto, johon kaikki löydetyt viat kerätään tarkastelua varten. Kuka tahansa projektissa työskentelevä tai tuotetta käyttävä henkilön pitäisi pystyä ilmoittamaan virhetilanteista. Raportoidut ongelmat voivat liittyä mihin vain, kuten esimerkiksi puutteelliseen oheistukseen, eikä vain ohjelmistovirheisiin. Tärkeintä on, että kaikki esiin tulleet ongelmat kirjataan ylös korjauspäätöksen tekoa varten. Aina kaikkea ei ole pakko korjata. Virheiden raportointi ei aina kulje vain testaajilta kehittäjille vaan jotkin virheraportit voivat olla virheellisiä, joissa testaaja tai käyttäjä on voinut väärin ymmärtää sovelluksen toiminnan. Tällöin kehittäjän on kommentoitava virhepalautetta ja varmistaa, että testaaja on ymmärtänyt ongelman oikein tai pyytää sen toistamiseen lisäohjeita. Testi raporttien tarkoitus on antaa testi- ja projektipäällikölle kokonaiskuva ongelmien määrästä ja niiden korjaustilanteesta. Tämän takia arkiston olisi hyvä sisältää työkalu, joka antaa tiivistetyn palautteen kokonaistilanteesta. Jotta raporttien käsitteleminen ja luominen olisi helppoa, tulisi niiden noudattaa yleistä kaava, joista löytyy kaikki tarvittava tieto omilta paikoiltaan. Tärkeitä tietoja ovat testattu ohjelma, ympäristö, testaajan nimi, ongelman kuvaus, toistettavuus ja arvio sen vakavuudesta. (11.)

5.5 Roolit testitiimissä

Selkeä työnjako ja vastuualueet ovat tärkeässä osassa toimivaa testauskokonaisuutta. Testaustiimi koostuu yleensä testipäälliköstä, testaajista ja automaation tasosta riippuen mahdollisesti myös automaatiosta vastaavasta pääkehittäjästä. Hierarkiassa testipäällikön rooli on näistä rooleista korkeimpana.

Testipäällikön työ on pääasiallisesti johtaa sekä koordinoida testauksen toimintaa ja kommunikoida siitä projektin johdolle. Tärkein testipäällikön tehtävistä on varmistaa, että kaikilla testaajilla on aina tarpeeksi tekemistä ja tekemisen kohde kohdistuu haluttuihin asioihin, että työnteko pysyy mahdollisimman tehokkaana. Yleensä tämä tapahtuu laatimalla testaussuunnitelman, jonka pohjalta testaajien työ voi alkaa. Testipäällikön

tehtävänä on myös järjestää ja huolehtia, että testaajilla on kaikki työhön tarvittavat resurssit ja työkalut saatavilla. Kommunikointi muiden tahojen kanssa on myös erittäin keskeisessä osassa testipäällikön työtä. Testauksen perimmäinen tarkoitus on kuitenkin saada tietoa tuotteen laadusta, jotta päätöksiä ja korjaustoimenpiteitä voidaan suorittaa. Jos kommunikaatio ei pelaa selkeästi ja nopeasti testauksen ja muun projektin välillä jää siitä saatava hyöty paljon pienemmäksi. Tämä johtuu pääasiallisesti siitä, että hitaan kommunikaation takia reagointi vikojen korjaukseen viivästyy ja näin ollen ohjelmissa esiintyvät viat kerkeävät aiheuttaa suuremman vaikutuksen. Testipäälliköltä vaadittava kommunikaatio testaustiimistä ulospäin ei kuitenkaan rajoitu vain pelkästään tulosten raportointiin, vaan on myös hyvin tärkeää tehdä ennalta ehkäisevää työtä. Tämä tapahtuu pääsääntöisesti ennakoimalla ja tunnistamalla mahdollisia ongelma ja riski kohtia koko projektin elinkaaressa. (8.)

Itse testaajien työkuva on suhteellisen suoraviivainen, heidän tehtävänään on suorittaa manuaalista sekä mahdollisesti automaatista hyväksymistestausta. Yleensä matalamman tason testaus, kuten yksikkö- ja integraatio testaus jätetään kehittäjien vastuulle. Testaajien työkuvaan kuuluu niin testitapausten suorittaminen, kuten myös virhetilanteiden mahdollisimman tarkka kuvaus. Virhetilanteiden kuvaukset yleensä sisältävät tiedon, milloin, missä ympäristössä, mahdolliset toimenpiteet ongelma toistamiseen ja testaajan nimen, jotta virheen mahdollinen jatkoselvittäminen olisi kehittäjälle mahdollisimman vaivatonta. (8.)

Isoimmissa ohjelmointiprojekteissa saattaa olla erilliset testaustiimit manuaaliselle ja automatisoidulle testaukselle. Näissä tilanteissa manuaalisen testauksen tiimin rakenne ja roolit pysyy täysin saman, kuin yllämainitussa kappeleissa. Automaatiotestauksen osalta tiimiin voidaan tarpeen mukaan valita johtava kehittäjä. Johtavan kehittäjän rooli vastaa hyvin pitkälti testipäällikön roolia, mutta on paljon painottuneempi teknisiin valintoihin.

6 Yleisimmät testaukseen liittyvät ongelmat projektissa

6.1 Aikataulupaineet

Testauksen yleisin ongelma on aikataulusta johtuvat kiireet. Kun testauksessa ajaudutaan tilanteeseen, missä aikataulut puskevat päälle, tarkoittaa tämä yleensä sitä, että testauksesta joudutaan tinkimään. Tästä syystä on erittäin tärkeää, että testauksen työ on hyvin priorisoitu ja suoritettu tärkeysjärjestyksessä. Jos testaus on jäänyt liian lyhyeksi aikataulusta johtuen, jää tuotteen omistajan vastuulle tehdä päätös tuotteen kohtalosta. Tällaisessa tilanteessa tuotteen omistajalla on oltava selkeästi tiedossa, mitä on ehditty testaamaan ja mitä ei, jotta hän voi arvioida, minkälaisia riskejä tuotteen julkaisemiseen kohdistuu. (10.)

Kaksi yleisintä syytä testauksen aikatauluongelmiin ovat liian kunnianhimoiset aikataulut ja kehitystyössä tapahtuneet viivästykset. Kehitystyön viivästyminen on hyvin yleistä ohjelmistoprojekteissa, koska yllättäviin ongelmiin ei yleensä osata varata tarvittavan paljon ylimääräistä aikaa. Tämä johtaa lopulta siihen, että tuote pääsee testaukseen suunniteltua myöhemmässä ajankohdassa, mutta tuotteen luvatusta julkaisupäivästä ei yleensä haluta luopua, joten testaus joutuu kärsimään tämän viivästyksen eron. (10.)

Tilanne pahentuu entisestään, jos testauksen aikatauluksi on jo valmiiksi luvattu liian lyhyttä aikaa. Tämä on hyvin yleistä silloin, kun testaus on ulkoistettu ulkopuoliselle tekijälle. Testauksen ulkoistaminen tapahtuu yleensä kilpailutuksen kautta, jolloin yritykset joutuvat lupaamaan melkein mahdottomia voittaakseen tarjouskilpailun. Tämän ratkaiseminen voi olla hyvin haastavaa, sillä valinnan tekevän tahon saattaa olla vaikea ymmärtää, miksi halvin eli nopeinten suoritettu testaus ole välttämättä se paras vaihtoehto. (10.)

Aikatauluja suunniteltaessa tulisi aina ajatella pidemmällekin tulevaisuuteen kuin kyseisen suunnitelman loppuun ja jättää ylimääräistä liukuma aikaa, että kehitys- ja testaustyö keretään tehdä kunnolla loppuun kulmia oikomatta. Kun aikataulu ongelmiin ajaudutaan kulmien oikominen kehityksessä ja testauksessa saattaa kuulostaa houkuttelevalta ratkaisulta ongelmiin, mutta todellisuudessa pidemmällä aikavälillä tästä saattaa koitua vain lisäkustannuksia, kun ohjelmistoa joudutaan korjaamaan jälkikäteen, joka on yleensä paljon hitaampaa ja haastavampaa kuin ensimmäisellä kehitys kerralla.

6.2 Testauksen irrallisuus

Testausprosessin irrallisuus kehitystyöstä aiheuttaa useita erilaisia ongelmia. Irrallisuudella tässä tapauksessa tarkoitetaan, että projektin testaustiimi ei sijaitse samassa tilassa muun kehitystyön kanssa. Suurin tästä seuraava ongelma on kommunikaation vaikeus ja hitaus. Sujuva ja selkeä kommunikaatio on erittäin tärkeää kehitys- ja testaustiimin välillä, sillä se helpottaa ja nopeuttaa ongelmatilanteiden ratkaisemista, mutta jos kommunikaatio joudutaan toteuttamaan sähköpostitse tai muuta viestintävälinettä käyttäen, on se todella tahmeaa. Samalla väärinymmärrysten mahdollisuus kasvaa, koska viestien tulkitsemiseen jää enemmän varaa, kuin toisen henkilön kanssa kasvokkain keskustellessa. Myös kynnys kysyä ja varmistaa asioita kasvaa, mitä vaikeampaa kommunikointi on. (7.)

Jo pelkästään toimivan kommunikaation takia testaus- ja kehitystiimit kannattaa sijoittaa mahdollisimman lähelle toisiaan, mieluiten jopa samaan tilaan. Näin kommunikaatio on mahdollisimman sujuvaa, kun kehittäjät ja testaajat voivat päivittäin keskustella keskenään ja sopia asioista. Tämä helpottaa myös ongelmatilanteiden ja mahdollisten virheiden selvittämistä, kun kehittäjä ja testaaja voivat vain istuutua saman tietokoneen ääreen ja selvittää asian, ilman ylimääräisiä sähköposti viestintää. Toimiva kommunikaatio mahdollistaa nopeamman asioihin reagoinnin, joka on erittäin tärkeää ketterässä ohjelmistokehityksessä. Tiivis yhteistyö eri osastojen välillä parantaa myös huomattavasti projektin ilmapiiriä, kun asiat sujuvat helposti ja joutuvasti. (7.)

6.3 Kokonaiskuva

Testauksen tavoitteena on pitää huoli, että ohjelmiston laatu pysyy määritellyllä tasolla. Joten selkeä kokonaiskuva ja tavoitteet ovat erittäin tärkeässä asemassa, kun korkeaa laatua pyritään ylläpitämään. Testauksen määritelmät tulee olla selkeästi dokumentoituna ja helposti saatavilla heti tarvittaessa. Jos testauksen toiminnassa on häiriöitä tai epäselvyyksiä, niin ei myöskään voida saada täyttä varmuutta testattavasta tuotteesta. Yleinen ongelma kokonaiskuvan suhteen tulee esiin etenkin, kun toimitaan ketterän ohjelmistokehityksen kanssa. Ketterä ohjelmistokehitys perustuu pieniin ja jatkuviin parannuksiin, esimerkiksi uuden version julkaisu viikoittain. Tällöin testauksenkin tulee pystyä reagoimaan muutoksiin erittäin nopeasti ja jatkuvasti. Nopea reagointi vaatii tarkan ja ennalta määritetyt toimintatavat, jotka pohjautuvat testisuunnitelmaan. Tällaisissa

projekteissa testiautomaatiorooli on suuressa osassa, koska jokaviikkoinen manuaalinen järjestelmätestaus vaatisi aivan liikaa resursseja. Kaikkea ei kuitenkaan ole mahdollista aina automatisoida, joten manuaalistakin testauta joudutaan aina suorittamaan. Tämän takia on erittäin tärkeää, että kokonaiskuva on aina ajan tasalla ja tieto manuaalisesti testattavista ominaisuuksista selvillä. Jos testaajien käsitys kokonaiskuvasta pääsee vääristymään tai vanhentumaa altistaa tämä testattavan tuotteen suuremmalle virheen mahdollisuudelle, kun tuotantoon vientiä edeltävä testaus jää puutteelliseksi. Siksi olisikin tärkeää aina aika ajoin muistaa pysähtyä tarkistamaan, että kaikki on prosessin osalta kunnossa, mutta tämä saattaa helposti unohtua jatkuvan kiireen ja uusien testaus kohteiden myötä. Näin projektissa voidaan ajautua tilanteeseen, että mahdolliset virheet tai riskit eivät ilmene uusissa ominaisuuksissa, niin kuin voisi olettaa vaan vanhoissa ominaisuuksissa regressiosta johtuvien muutosten myötä. (9.)

Lähteet

1. Software testing, An ISTQB-ISEB Foundation Guide Second Edition, Brain Hamblin, BCS the chartered institute for IT.

2. Software testing foundation 2nd edition, Andreas Spiller, Tilo Linz, Hans Schaefer, Rocky nook Inc.

3. VAAKA-hanke, Väestörekisterikeskus, Retro 13.03.2019, Aihe: Aurinkoista & kyyneliä.

4. VAAKA-hanke, Väestörekisterikeskus, Retro 10.04.2019, Aihe: Yleinen, miten sprintti on mennyt.

5. VAAKA-hanke, Väestörekisterikeskus, Retro 08.05.2019, Aihe: Ulkoisten integraatioiden ongelmat.

6. VAAKA-hanke, Väestörekisterikeskus, Retro 22.05.2019, Aihe: Testien toimivuus yleisesti.

7. VAAKA-hanke, Väestörekisterikeskus, Retro 19.06.2019, Aihe: Yleisiä onnistumisia ja ongelmia.

8. VAAKA-hanke, Väestörekisterikeskus, Testausprosessin jatkokehittäminen 16.09.2019.

9. VAAKA-hanke, Väestörekisterikeskus, Testausprosessin jatkokehittäminen 7.10.2019.

10. VAAKA-hanke, Väestörekisterikeskus, Testausprosessin jatkokehittäminen 23.9.2019.

11. VAAKA-hanke, Väestörekisterikeskus, Testausprosessin jatkokehittäminen 30.9.2019.

12. VAAKA-hanke, Väestörekisterikeskus, Retro 25.09.2019, Aihe: Hyvät, pahat ja rumat.

13. VAAKA-hanke, Väestörekisterikeskus retro 28.08.2019, Aihe: Vapaa.